Aalto University
School of Science
Master's Programme in Security and Mobile Computing

Martin Gomez Gonzalez

Operator authentication and accountability for SCADA servers when requests are forwarded by a middle layer

Master's Thesis

Espoo, June 1, 2018

Supervisors: Tuomas Aura, Professor
Prof. Mathias Ekstedt (KTH, Sweden)

Thesis advisor(s): Pontus Johnson, Professor (KTH, Sweden)

AALTO UNIVERSITY
School of Science
Master's Programme in Security and Mobile Computing

ABSTRACT OF THE
MASTER´S THESIS

Author: Martin Gomez Gonzalez

Title of the thesis: Operator authentication and accountability for SCADA servers when requests are forwarded by a middle layer

Number of pages: 93

Date: 01/06/2018

Major or Minor
Security and Mobile Computing

Supervisor: Prof. Tuomas Aura and Prof. Mathias Ekstedt (KTH, Sweden)

Thesis advisors: Prof. Pontus Johnson (KTH, Sweden)

Due to their critical nature, the actions performed by operators on Industrial Control Systems (ICS) are subject to source authentication and accountability. When commands are not send directly by the user, but forwarded by middle servers, the compromise of those severs threatens the security of the whole architecture. This Master thesis provides a solution for that problem, guaranteeing authentication end-to-end while fulfilling cost and performance requirements. Based on an analysis of several potential solutions, digital signatures were assessed to be the most flexible and secure option. Moreover, the proposed solution relies on Microsoft's Active Directory, which manages credentials on the target architecture, for securely linking public keys with user identities. A prototype implementation of the proposed design is included, together with a limited performance evaluation. They have proven the validity of the design, that guarantees end-to-end authentication and accountability of command requests, while maintaining low implementation and maintenance costs and a negligible impact in latency per message.

Keywords: SCADA, ICS, authentication, non-repudiation, digital signatures, Active Directory, asymmetric cryptography, ECDSA, RSA

Publishing language: English

Contents

1 Introduction **1**

1.1 Problem Description 1

1.2 Objectives 1

1.3 Research Question 2

1.4 Delimitations and scope 2

1.5 Contributions 2

1.6 Thesis outline 3

2 Theoretical foundation **4**

2.1 Authentication 4

2.2 Non-repudiation 4

2.3 Symmetric-key and Public-key cryptography 4

2.4 Public-key algorithms 5

2.4.1 RSA 5

2.4.2 ECDSA 5

2.4.3 Performance of RSA and ECDSA 7

2.4.4 Security of RSA and ECDSA 8

2.4.5 Cryptographic hash functions 10

2.5 Cryptographic solutions for authentication and non-repudiation . 10

2.5.1 Digital signatures 10

2.5.2 Keyed-Hash Message Authentication Code (HMAC) . . . 11

2.5.3 Symmetric key based solutions 12

3 Background **16**

3.1 Industrial Control Systems 16

3.2 SCADA constrains 17

3.3 SCADA security procedures 19

3.4 Multi-tier authentication 20

4 Methodology **22**

5 Requirements **25**

5.1 Architecture constrains 25

5.2 Security requirements 26

5.3 Performance requirements 26

6 Analysis **28**

6.1 Trusting the access server: Kerberos delegation 28

6.2 Symmetric-key cryptography: HMAC over the current Kerberos architecture 29

6.2.1 Unknown server HMAC-based authentication end-to-end 31

6.2.2 Unknown server HMAC-based authentication with access server implication 32

6.2.3 HMAC-based authentication for a pool of servers 32

6.3 Public-key cryptography: Digital signatures over the current Kerberos architecture 33

6.3.1 Public Key Infrastructure for credential management . . . 33

6.3.2 Active Directory for credential management 35

6.4 Discussion 37

6.5 Conclusion 41

7 Design **43**

7.1 Message authentication and integrity 43

7.1.1 Signing 44

7.1.2 Verification 45

7.2 Credential management 46

7.2.1 Credential management server and schema modifications . 46

7.2.2 Signing node 47

7.2.3 Verifying node 48

7.3 Requirement fulfilment 48

7.3.1 Architecture constraints 48

7.3.2 Security requirements 49

7.3.3 Performance requirements 50

8 Implementation **51**

8.1 AD Server 51

8.1.1 Adding a field for storing operator public keys 51

8.1.2 Writing permissions of the public key field 52

8.1.3 Reading permissions of the public key field 52

8.1.4 Purging obsolete keys 52

8.1.5 Domain controller replication 53

8.2 Signing node 54

8.2.1 Active Directory connection (Login) 55

8.2.2 Socket client (Connect) . . . 57
8.2.3 Signature (Messaging) . . . 58
8.3 Verifying node . . . 59
8.3.1 Key retrieval from the AD . . . 60
8.3.2 Key parsing . . . 60
8.3.3 Validation of signed messages . . . 61

9 Performance evaluation 62
9.1 Cryptographic operations . . . 62
9.1.1 Key generation (Signing node) . . . 62
9.1.2 Signature generation (Signing node) . . . 63
9.1.3 Signature verification (Verifying node) . . . 63
9.2 Message passing . . . 63
9.3 Key retrieval delay . . . 64

10 Discussion 65
10.1 Compatibility with the target architecture . . . 65
10.2 Security . . . 65
10.2.1 Digital signatures . . . 65
10.2.2 Active Directory for identity management . . . 66
10.2.3 Peculiarities of the target system . . . 66
10.2.4 Additional security considerations . . . 67
10.3 Performance . . . 67
10.3.1 Cryptographic algorithms . . . 68
10.3.2 Key retrieval delay . . . 69

11 Conclusion 71
11.1 Future work . . . 71

Glossary 77

Acronyms 78

A Installation manual for the Active Directory Domain Controller 81

List of Figures

1 Symmetric-key encryption [1] 5
2 Asymmetric-key encryption [2] 5
3 Kerberos authentication process [3] 13
4 Single-hop and multi-hop authentication 20
5 Design Science Research Cycle [4] 22
6 Simplified representation of the current architecture 25
7 Message with HMAC sent to the access server 30
8 Login form of the signing application 55
9 Key management classes of the signing application 55
10 Common public key format 56
11 Public key format (ECDSA with curve NIST P-256) 56
12 Public key format (RSA 1024 bit) 57
13 Public key format (RSA 1024 bit): PUBLICKEYSTRUCT . . . 57
14 Public key format (RSA 1024 bit): RSAPUBKEY 57
15 Connect form of the signing application 58
16 Messaging form of the signing application 59
17 Opening the delegation wizard for the *User* container in the *test.m* domain 83
18 *Delegation of Control Wizard*: *SELF* user added for delegation . 83
19 *Delegation of Control Wizard*: Create a custom task to delegate . 84
20 *Delegation of Control Wizard*: Selecting the *User* object 84
21 *Delegation of Control Wizard*: Selecting the field that stores the keys 85
22 Advanced security options the *Users* container in the *test.m* domain 87
23 *Users* container *allow* policy 88
24 *Users* container *Read thesisOperatorPublicKeys* 88
25 *test* domain container *deny* policy 89
26 *test* domain read and write *thesisOperatorPublicKeys* 89
27 Active Directory Sites and Services IP link 91
28 Active Directory Sites and Services: Switching *USE_NOTIFY* on 91
29 Active Directory User and Computers: User container auditing rules 92

List of Tables

1 RSA key generation 5

2 RSA encryption and decryption 5

3 Comparison of key sizes: Asymmetric (RSA and ECC) and symmetric. Based on to two publications [5][6]. 7

4 Performance comparison of asymmetric signature algorithms. Values are the median of the results of several executions, expressed in thousands of processor cycles [7]. A star indicates unreliable results (high variance). Implementation notes at [8]. 8

5 Comparison of the impact of the proposed solutions from a technological perspective. Each value is not absolute, but given in relation to the rest of the solutions 39

6 Comparison of proposed solutions depending from a deployment cost perspective. Each value is not absolute, but given in relation to the rest of the solutions 40

7 Testing system technical specifications 62

8 Key generation time in milliseconds (10000-element sample) . . . 63

9 Message hashing and signing time in milliseconds for 100-byte messages (10000-element sample) 63

10 Message hashing and verification time in milliseconds for 100-byte messages (1000-element sample) 63

11 Active Directory public key propagation times 64

12 Public Keys script parameters 82

1 Introduction

Due to the real-time nature and the importance of industrial processes, the remote operation of them is a requirement by many organizations. However, the ubiquity of Internet access is leading to an increment on the number of connections, reaching loads that SCADA servers may not be prepared to manage. A middle layer of access servers can be used to manage this connection load and forward the petitions to their final destination server(s).

Nevertheless, with the presence of a middle layer, authentication node-to-node does not guarantee authenticity and non-repudiation of the commands received at the SCADA server. Should one of the middle servers be compromised, the authentication they provide would not be trustworthy. Thus, this thesis intends to find a solution that guarantees end-to-end authentication, even in the event of access server compromise.

1.1 Problem Description

Operators may need remote access to SCADA[1] systems to check their status or run commands. In order to provide scalability and flexibility, a middle layer of access servers can be used to forward the petitions to the SCADA servers, reducing the number of direct connections to them.

This layer of access servers also provides a means of transparently redirect operator petitions to additional servers (e.g. historian server), leaving room for extending system functionalities without modifications on operator applications.

Moreover, complex commands may need actions to be performed in several SCADA servers. Access servers can receive a single request from an user and translate it into several commands to be send to various SCADA servers.

Nevertheless, when this middle layer between the SCADA server and the operator is in place, new challenges arise. Authentication of users and accountability of their actions become a more complex issue. Particularly, there is a need for the SCADA server to know who initiated each command. At the same time, the access server needs to know if the user is authenticated for performing the requested action, to decide whether it should forward the request to the SCADA server.

In addition, the complexity should not be handed over to the user (i.e. they should not need to log in twice), but the system must manage the double authentication transparently. Finally, SCADA systems have some requirements in terms of latency and protocol compatibility that should be considered.

1.2 Objectives

This project aims to provide a method for securing the access to the SCADA servers through a middle layer, ensuring authentication and accountability at each step of the communication. To this effect, several solutions will be explored

[1] **SCADA**: Computer based system architecture for management and supervision of industrial processes.

and evaluated in terms of security, cost, complexity and compatibility with an existing deployment at ABB.

Moreover, the prototype implementation of the proposed design is also to be delivered by this thesis. It aims to prove the validity and suitability of the design and to provide a test bed for running a limited performance evaluation of the solution.

1.3 Research Question

This Master thesis aims to answer the following research question:

> What is the best method to provide authentication and accountability on the communication between a SCADA server and a remote operator when an access server is in the middle?

Two aspects of this question are to be considered:

- Theoretical view: Which option better satisfies the requirements and offers a higher level of security.
- Practical view: Which option that meets the requirements is the best in terms of cost, latency and compatibility with the existing set up.

1.4 Delimitations and scope

The main focus of this thesis lies in defining the design of a solution that addresses the research problem (§1.1), while achieving the objectives (§1.2). Thus, the design must be targeted to ABB's architecture and requirements.

However, architecture parameters vary from one ABB client to another. Thus, a concrete set of requirements must be established, agreeing with ABB in the general characteristics of their client's architectures. Having a concrete set of requirements may limit the compatibility of our design to a subset of all ABB's client architectures, but is needed for delimiting the thesis work.

Moreover, our prototype implementation is not generated to be deployed in production environments, but for proving the validity of the design and as test bed for the performance evaluation. Therefore, the main focus is to be able to sign and verify messages, using the Active Directory for credential management.

The actual message exchange between nodes depends on several ABB RPC protocols and is not necessary for the objectives of this Master thesis that the implementation supports them. It is sufficient to use a means of communication that is compatible with these protocols (i.e. text-base communication) to ensure that our design can be implemented on top of them.

1.5 Contributions

The main contribution of this Master thesis is a design that provides the best solution for securing the communication between SCADA servers and operators

through a middle layer, considering the particular requirements of the target architecture.

Moreover, the analysis of several solutions prior to the design proposal has value in its own, since it proposes several valid solutions to the problem that may be more suitable than the selected one for different target architectures.

In addition, a prototype implementation and a limited performance evaluation is also provided, facilitating the implementation of the solution in production environments.

Furthermore, we define how to use the Active Directory for storing public keys. This aspect of the implementation is valuable on its own, as it can be used for storing public keys in other architectures and with different purposes.

1.6 Thesis outline

This thesis report is organized in the following chapters:

1. **Introduction**: This chapter introduces the topic of this thesis by defining the problem and how is addressed.
2. **Theoretical foundation**: It introduces several theoretical concepts that this Master thesis builds upon.
3. **Background**: It extends the theoretical foundation by covering a set of definitions that are especially relevant for this thesis work.
4. **Methodology**: It discusses about the methodology followed for producing this Master thesis.
5. **Requirements**: It establishes the requirements that the solution to be proposed by this thesis must fulfil.
6. **Analysis**: It discusses the possible designs that would fulfil the previously established requirements.
7. **Design**: It provides a detailed description of the selected solution.
8. **Implementation**: It describes the creation of a prototype implementation of the design
9. **Performance evaluation**: It covers the results of a limited performance evaluation of the implementation.
10. **Discussion**: It explains how the thesis requirements are met in terms of compatibility with the target architecture, security and performance.
11. **Conclusion**: It reflects about the level of fulfilment of the thesis objective and proposes potential extensions.

2 Theoretical foundation

This chapter introduces several theoretical concepts that this Master thesis builds upon. Thus, it reflects the research that supports the rest of the chapters and provides a means for understating the thesis to the readers.

2.1 Authentication

In computing, authentication is the process that ensures that an entity is who it claims to be. When a user sends messages or accesses a resource, authentication guarantees that their identity is verified. This process is needed, not only for guaranteeing the validity of the user's identification, but also for supporting other security related processes. Namely, authorization and non-repudiation. Users' identities should be verified before deciding which permissions they have or to have a reliable record of their accesses to the system.

2.2 Non-repudiation

Non-repudiation in electronic transactions refers to the impossibility for the involved parties to deny their participation. For the target architecture, the focus is on the user actions. The SCADA server needs to guarantee that when an user issues a command, the action is registered and the user cannot deny having sent that command to the server. Thus, in our context, non-repudiation has the same effect as accountability. Providing the system guarantees user authentication, if it ensures accountability (i.e. it records user identities and their actions) it also ensures non-repudiation (i.e. the user cannot deny issuing a command).

2.3 Symmetric-key and Public-key cryptography

Modern cryptographic algorithms can be classified in two groups, depending on the type of encryption algorithms they use. Symmetric key cryptography relies on one key that is used for encrypting and decrypting messages, as shown in Figure 1. Therefore, anyone with the secret key can perform encryption and decryption operations.

On the other hand, when asymmetric or public key encryption is in place, each entity has two keys: a public and a private one. Messages are encrypted with the public key of the recipient who decrypts them with their private key. The public key is used for encrypting messages and the private key for decrypting them. The process is show in Figure 2

However, public key encryption algorithms have a higher computational cost than symmetric ones. Mainly because their keys are longer. They need a greater protection against brute-force attacks, since public keys are expected to be more exposed than symmetric keys. Thus, symmetric keys are typically used for perform bulk encryption operations, while public-key based algorithms provide a secure method for exchange symmetric keys, which are typically temporary for a communication session.

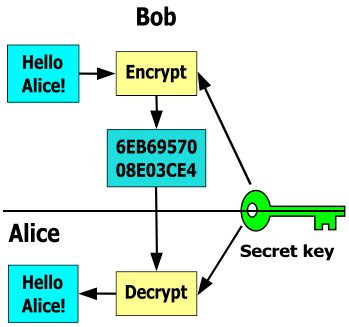

Figure 1: Symmetric-key encryption [1]

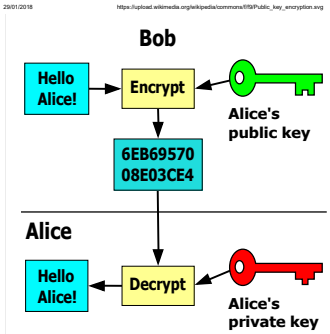

Figure 2: Asymmetric-key encryption [2]

Table 1: RSA key generation

Step	Algorithm	Example
1	Select p and q, both prime and $p \neq q$	$p = 17$ and $q = 13$
2	Obtain n: $n = p \cdot q$	$n = 17 \cdot 13 = 221$
3	Obtain $\phi(n)$: $\phi(n) = (p-1) \cdot (q-1)$	$\phi(n) = (17-1) \cdot (13-1) = 192$
4	Obtain e: $\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$	$e = 11$; $\gcd(192, 11) = 1$; $1 < 11 < 192$
5	Obtain d: $d \cdot e \mod \phi(n) = 1$	$d = 35$; $35 \cdot 11 \mod 192 = 1$
6	Public Key: $PU = \{e, n\}$	$PU = \{11, 221\}$
7	Private key: $PV = \{d, n\}$	$PV = \{35, 221\}$

2.4 Public-key algorithms

2.4.1 RSA

Rivest, Shamir and Adleman (RSA) is the most widely used asymmetric key algorithm [9]. Its design is based on the unproven assumption that the factorization of the product of two large prime numbers is too difficult to be done in a practical period of time. The key generation process of RSA is show in Table 1. Moreover, as presented in Table 2, RSA supports both encryption and digital signature operations, depending on how the keys are used.

2.4.2 ECDSA

Digital Signature Algorithm (DSA) is a digital signature standard defined by the US National Institute of Standards and Technology [10]. Unlike RSA, it

Table 2: RSA Key encryption and decryption.[2]

Action		Typical usage	Algorithm
Encryption	Public Key	Confidentiality: message encryption	$C = M^e \mod n$
	Private Key	Authentication: signing	$SM = M^d \mod n$
Decryption	Public Key	Authentication: signature verification	$M = SM^e \mod n$
	Private Key	Confidentiality: message decryption	$M = C^d \mod n$

does not support encryption operations. Moreover, its design lies on discrete logarithms and how difficult they are to compute, compared to their inverse operation, the discrete exponentiation. As for RSA, this assumption is not proven.

Discrete logarithm cryptography can be divided into finite field cryptography, the one used by DSA, and elliptic curve cryptography. The latter is used by Elliptic Curve Digital Signature Algorithm (ECDSA), an algorithm that is replacing DSA at several organizations. In fact, the NSA (US National Security Agency) does not recommend DSA anymore, but they suggest relying on RSA or ECDSA [11] for digital signatures. The main advantage of ECDSA, when compared to DSA and also to RSA, is that it provides the same level of security using shorter key lengths.

An elliptic curve is a set of points defined over a finite field $\mathbb{F}_q$. An specific curve E is the set of solutions to its short Weierstrass equation.

If $\mathbb{F}_q$ is a prime field ($\mathbb{Z}_p$), E is defined by [12]

$$E(\mathbb{F}_q) = E(\mathbb{Z}_p) = \big\{(x, y) \in \mathbb{Z}_p^2 \mid y^2 = x^3 + ay + b\big\},$$

where $(a, b) \in \mathbb{Z}_p$ and $4a^3 + 27b^2 \neq 0$.

However, when $\mathbb{F}_q$ is a binary field, the equation that defines E is [12]

$$E(\mathbb{F}_q) = E(\mathbb{F}_{2^l}) = \big\{(x, y) \in \mathbb{F}_{2^l}^2 \mid y^2 + xy = x^3 + ax^2 + b\big\},$$

where $(a, b) \in \mathbb{F}_{2^l}$, $b \neq 0$ and 2^l is a power of 2.

Key generation ECDSA keys are relative to a particular elliptic curve, defined by the following public parameters ($s \times P$ represents an elliptic curve point P multiplied by an scalar s) [13]:

- q, the cardinality of the curve field, $\mathbb{F}_q$
- a and b, two field elements
- n, being n prime and $n > 2^{160}$
- G, an element of the elliptic curve of order n ($n \times G = 0$)

With these parameters, the key pair can be generated. The private key is a random integer $d \in [1, n-1]$. The public key is $d \times G$.

Signing As with RSA or DSA, signature operations are done using the hash of the document, $H(M)$. The particular steps for generating a signature from the document hash are detailed below [13]:

1. Select k, a random integer. $k \in [1, n-1]$.
2. Obtain a curve point: $(x_1, y_1) = kG$

[2]C=Ciphertext; M = Plaintext; SM = Signed message

Table 3: Comparison of key sizes: Asymmetric (RSA and ECC) and symmetric. Based on to two publications [5][6].

Security (bits)	**Symmetric**	**Asymmetric** [5]		**Asymmetric [6]**	
		RSA	**ECC**	**RSA**	**ECC**
112	3TDEA	2048	224-255	2432	224
128	AES-128	3072	256-383	3248	256
192	AES-192	7680	384-511	7936	384
256	AES-256	15360	≥ 512	15424	512

3. $r = x_1 \mod n$. If $r = 0$ go back to 1.
4. $s = \frac{H(M)+d \cdot r}{k} \mod n$. If $s = 0$ go back to 1.
5. The signature of message M is (r, s).

Signature verification Finally, when the recipient obtains the signed message, they need to verify the signature to ensure the integrity of the message (i.e. it was not modified in transit) and to authenticate the sender to be the owner of the public key, ensuring that the following conditions are true [13]:

- $Q \neq 0$, $Q \in C$ and $n \times Q = 0$
- $r, s \in [1, n-1]$
- $r = x_1 \mod n$ for:

$$(x_1, y_1) = u_1 \times G + u_2 \times Q$$
$$u_1 = \frac{H(M)}{s}$$
$$u_2 = \frac{r}{s} \mod n$$

2.4.3 Performance of RSA and ECDSA

The performance of any cryptographic algorithm varies according to its key size. Greater key sizes usually imply more complex operations and a resulting increase on the computation time for any cryptographic operations. This is generally true for RSA and for Elliptic Curve Cryptography (ECC). However, with ECC, the particular choice of curve can lead to greater key lengths being faster. For instance, both signing and signature verification operations are faster with the NIST P-256 (256 bits) curve than with the NIST P-224 (224 bits) [7].

Moreover, any performance analysis must acknowledge that ECC (and thus ECDSA) provides the same level of security with smaller key sizes, as Table 3 shows.

Table 4 shows the number of cycles for digital signature related operations depending on the algorithm and key-size. As previously stated, the NIST P-224 curve offers worse performance that the NIST P-256 while providing a lower level of security. Thus, for ECDSA, only the 256 bits curve is considered, and compared with RSA 3072 (equivalent) and RSA 2048.

Table 4: Performance comparison of asymmetric signature algorithms. Values are the median of the results of several executions, expressed in thousands of processor cycles [7]. A star indicates unreliable results (high variance). Implementation notes at [8].

Primitive	Key generation	Signing (56 bytes)	Verification (56 bytes)
ECDSA NIST P-256	493	163	311
RSA 2048-bit	167120*	3363	51
RSA 3072-bit	576112*	8518	85

According to those benchmarks (on Table 4), key generation is 3 orders of magnitude faster for ECDSA than for RSA. Theory supports this result. RSA key generation includes the lookup of large prime numbers. There are known processes for doing so, but they are computational expensive.

On the other hand, ECC relies on a set of pre-established parameters for defining the curve. The generation of they key pair only needs the generation of a random number (the private key) and obtaining a point on the curve by multiplication. As a result RSA key generation times grow exponentially, while with ECDSA the increment is linear in relation with key size [14].

Moreover, in terms of signature generation, ECDSA is one order of magnitude faster than RSA. The reason behind this difference is that ECDSA signing relies on the Extended Euclidean Algorithm, which is more computationally efficient that the exponentiation needed by RSA [15]. Results from other performance studies support this consideration [16][17]. Other analysts report differences only for big key sizes [14].

Finally, signature verification is 1 order of magnitude faster when using RSA than with ECDSA. The theoretical explanation is that RSA only needs one exponentiation operation for verifying a signature, while ECDSA needs two. This fact is supported by several performance studies [16][17][14].

2.4.4 Security of RSA and ECDSA

Both ECDSA and RSA provide a great level of security, providing that keys are long enough. The United States National Security Agency (NSA), establishes RSA as suitable for *TOP SECRET* systems as long as keys are at least 3072 bit long. For ECDSA, they require 384-bit keys, recommending the NIST-P-384 curve [11].

The ECRYPT group has similar recommendations. According to the them, RSA 2048-bit and ECDSA 224-bit are suitable for medium term protection while long term protection can be guaranteed with RSA 3072-bit or ECDSA 256-bit [6].

In general, any public key algorithm must rely on a one-way function. It must be impossible to obtain the private key or simulate its ownership, even if the public key is known.

As stated before, RSA guarantees this property by using prime number factorization, while ECDSA or DSA are based on the discrete logarithm problem. Both operations are supposed to be too computationally expensive to revert

when key lengths are sufficient, but this theory is not proven.

Moreover, quantum computing could be able to break both RSA or ECDSA. In 1999, Peter W. Shor [18] presented an algorithm to solve both prime factorization and discrete logarithms on a quantum computer in polynomial times. However, no organization or individual has acknowledged having broken RSA or ECDSA using quantum computing.

There are other public-key cryptography algorithms that are supposed to be resilient to quantum computing, such as Lattice-based or hash-based cryptography [19], but they are not widely used or standardised [11].

RSA vulnerabilities Due to its wide use and deployment, RSA has been extensively tested. There are optimizations that can solve the factorization problem in shorter periods of times. The longer RSA key that has been broken is a 768-bit one. The researchers needed several years to accomplish this milestone. Consequently, they recommended using RSA keys longer that 1024 bits [20].

Moreover, other attacks against RSA have focused on implementation problems. For instance, vulnerabilities in the pseudo-random number generators [21] or defective key generation (p and q values being to close) [22]. In addition, side channel analysis attacks are also possible. For example, time based attacks that rely on the knowledge of the victim's hardware [23].

ECDSA vulnerabilities The main disadvantage of ECDSA in relation with RSA is its more recent deployment. Even though ECDSA is becoming to be widely used, its implementations have not been as thoroughly tested as RSA ones. As a result, the likeliness of undiscovered or undisclosed vulnerabilities is arguably higher. Moreover, ECDSA is more difficult to implement [24], which makes harder for developers to evaluate existing implementations and new ones more bug prone.

As for RSA, ECDSA implementations could be faulty and have vulnerabilities. One important parameter of ECDSA (and classic DSA) is k. If its value is repeated along several uses of the private key, and attacker could be able to obtain the key. As a result, k should be a nonce (only used once) and its generation unpredictable and secret [25][26]. One alternative for guaranteeing that k is unique and unpredictable is to deterministically generate it based on the private key and the message hash, as described in RFC 6979 [27].

In addition, the selection of a particular curve can have security implications. Some curves are considered insecure, since they reduce the computational cost of the related discrete logarithm operation [28]. A solution to this potential flaw is using known curves. The most commonly used ones are the curves standardised by the United States National Institute of Standards and Technology (NIST). However, it should be noted that some cryptography experts argue that these curves are not completely secure. They support this allegation on the use of unexplained seeds for generating coefficients [29].

Finally, attackers could also benefit from knowledge or access to the victim's system to perform side-channel attacks, in a similar manner as it has been done for RSA. These attacks could even be performed remotely, through a network

service [30].

2.4.5 Cryptographic hash functions

A hash functions map data of an arbitrary length to a fixed length data block. They are a digest of the data, typically used to ensure its integrity.

Hash functions are commonly used in cryptography. However, not any hash function is valid for this application. Cryptographic hash functions are those that are suitable for cryptographic operations. They have the following characteristics [9]:

- Deterministic: One input data block must always provide the same digest.
- They accept any input length.
- The output has a fixed length.
- The generation of the hash of a data block, $H(x)$, does not incur in high computational costs.
- One-way or preimage resistant: It is not computationally possible to recover a message m from its hash $H(m)$.
- Weak collision resistant: For a given data block x it is not computationally possible to find y such that $H(y) = H(x)$ and $x \neq y$.
- Strong collision resistant: It is not computationally possible to find a two data blocks (x, y) such that $H(y) = H(x)$ and $x \neq y$.

2.5 Cryptographic solutions for authentication and non-repudiation

2.5.1 Digital signatures

Digital signatures are the most common cryptographic solution for ensuring authentication and non-repudiation. They are based on asymmetric key encryption and they also guarantee the integrity of the transmitted data. Any of the previously mentioned public-key algorithms (RSA, DSA or ECDSA) can be use for implementing digital signature architectures.

> Digital signature schemes allow a signer S who has established a public key **pk** to "sign" a message in such a way that any other party who knows **pk** (and knows that this public key was established by S) can verify that the message originated from S and has not been modified in any way. [31]

When digitally signing a document or a message, the related cryptographic operation is not performed on the actual content to be signed, but on a digest of it. By doing so, the related computational costs decrease, together with the size of the signature. Moreover, a digest of the data provides a summary of its

contents in a standard format, simplifying the implementation of the signature. It also removes the need for signing several blocks when the input is longer than the block size.

The digest of the data needed for digital signatures must be done using cryptographic hash functions. Using a secure hash function ensures that the signature is valid for the messaged whose hash was signed.

MD5 and the SHA family have been the most common secure hash functions. However, MD5 and the initial versions of SHA (SHA-0 and SHA-1) are no longer considered secure [32][33]. The different versions of SHA-2 are the current NIST standard for digital signatures and they are currently working on SHA-3 [34]. They particularly recommend using SHA-2 hashes of at least 256 bit (SHA-256) or switching to SHA-3 [35]. The NSA requirements are higher, their advice is to use SHA-384 [11].

Digital signature security Digital signature security is dependant on the algorithms used for implementing it. As stated before, public key algorithms security typically depends on mathematical problems that are allegedly impractical to solve. Moreover, faulty implementations of these algorithms or of the digital signature process itself could lead to vulnerabilities.

Furthermore, public key authentication relies on the use of certificates, which are issued by a Certification Authority (CA), that manages a Public Key Infrastructure (PKI) environment. As a result, if the CA was compromised, an attacker would be able to impersonate any user by creating fake certificates.

Moreover, PKI environments are primarily stateless. The certificate is the main source of trust. Thus, if an attacker managed to obtain a legitimate certificate, they would be able to impersonate the owner.

For addressing the risk of stolen credentials, certificates have expiration dates and deployments can support the revocation of certificates. Revocation is provided by Certificate Revocation Lists (CRLs), a list of revoked certificates that should be periodically fetched from the CA, or by using the Online Certificate Status Protocol (OCSP), a protocol to verify with the CA the validity of a particular certificate. The latter provides a faster revocation, since certificates are verified on demand, but it also imposes an increment in latency for each certificate verification that CRLs do not.

2.5.2 Keyed-Hash Message Authentication Code (HMAC)

A Message Authentication Code (MAC) is a fingerprint of a message used to authenticate its origin, protecting data integrity and authenticity. In contrast with digital signatures, MACs are generated and verified using the same key.

One common implementation of MAC is HMAC, which uses a secret key and a hashing algorithm for generating the MAC. It is defined in RFC 2104 [36] as

$$HMAC(K, m) = H\big(K \oplus opad \| H(K \oplus ipad \| m)\big)$$

where K is the key, m is the message, $H(x)$ is the hash digest of x, *opad* and *ipad* are padding values, $\oplus$ is an XOR operation and $\|$ is a byte concatenation

operation.

Using an HMAC implementation provides flexibility since it can be used regardless of the underneath hashing algorithm. In addition, it provides a higher degree of security than more simple options such as $H(K\|m)$, $H(m\|K)$ or even $H(K\|m\|K)$, vulnerable to extension or collision attacks [37].

Security of HMAC is highly dependant on the length of the secret key, since it makes the digest vulnerable to brute force attacks, which when successful would provide the secret key to an attacker. On the other hand, collisions are not as critical as with digital signatures (the implementation was designed to mitigate their effect). Therefore, MD5 or SHA1 collision attacks do not seriously affect HMAC. However, more secure hash functions (at least SHA-256) should be used whenever possible, as recommended by the informational RFC 6151 [38] and NIST [35].

2.5.3 Symmetric key based solutions

The sole use of symmetric-key cryptography does not provide full authentication as is. In any pure symmetric-key protected communication, the key must be known at least by the parties who are involved. As a result, it can only unequivocally identify a sender if the recipient is fully trusted. Moreover, pure symmetric-key authentication is not manageable, since, to ensure security, each client would need to have a different key for each principal they want to communicate with.

In order to address this limitation, the use of authentication servers is needed. An authentication server is a trusted party that authenticates all the nodes involved in a electronic transaction by using symmetric-key cryptography and provides the tools for them to authenticate against each other. The most common protocol to perform these operations is Kerberos.

Kerberos protocol main functionality is the verification of principals (e.g. users or servers) on an insecure network. It does not depend on physical security, host addresses (e.g. IP addresses) or host operating systems, and eavesdropping of the packets sent through the network is assumed. [39]. A simplified description of the authentication process is included bellow:

1. A client requests credentials for a particular server to the Authentication Server (AS)
2. The server response includes a ticket for the server and a (temporary) session key, encrypted with the client's (symmetric) key.
3. The client sends the ticket[3] to the server
4. Upon reception of the ticket, both the server and the client have the session key, which proves their identities were verified by the AS.

Typically, the process is more complex. The AS does not send a client-to-server ticket directly, but a Ticket Granting Ticket (TGT), which has a similar

[3] A client-to-server ticket is formed by the client's identity and the session key, and encrypted with the server's key

structure. With the TGT, the client can request client-to-server tickets during a certain time period. Requests are sent to the Ticket Granting Server (TGS), which can be hosted by the same machine as the AS or by another one, providing scal

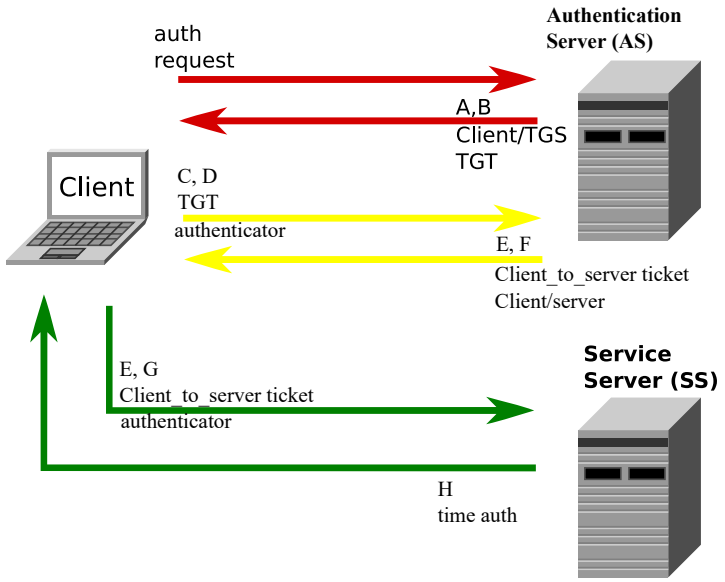

Figure 3: Kerberos authentication process [3]

Kerberos security Kerberos protocol security relies on the use of tickets and on the trust in the authentication server. Being security mainly based on tickets, means that Kerberos deployments have an inherent stateless functioning. Thus, tickets are portable, contain all the policy information and their protection must be a priority for any Kerberos implementation.

As a result, ticket encryption is not done directly with the account password, but with a derived key, from which the password cannot be recovered. Using derived keys makes harder for attackers to obtain the account password.

According to the Kerberos standard, keys should be created using a one-way function [39][40]. This (hash) function should not only receive the password, but also extra data (e.g. username and domain). Passing this data to the hash function avoids that users with the same password have the same derived keys.

However, Microsoft implementation used to ignore this standard requirement, by using Microsoft NT LAN Manager (NTLM) hashes directly to encrypt Kerberos tickets. NTLM hashes only have the user's password as input, and in NTLMv2 they are created using MD4, a protocol known for not being collision resistant [41]. This flaw was fixed in most recent Microsoft's implementations [42], that use AES-128 or AES-256 as the preferred encryption algorithms and whose key material includes not only the password but also the the user name and the DNS realm.

Typical attacks to the Kerberos protocol As stated previously, Kerberos security is primarily based on the use of tickets. Thus, most attacks try to target them, being therefore able to impersonate legitimate users. A summary of most common attacks against Kerberos protocol is shown below:

- **Overpass-the-hash** If attackers got their hands on the hash of an user account (e.g. from memory, by cracking a ticket), they would be able to request the Key Distribution Center (KDC) valid tickets for that user, thus being able to impersonate them [43]. This attack can be mitigated by using the latest Microsoft Kerberos implementation, that, as stated before, provides higher security in terms of password hashing and management.
  - **Golden ticket** If an attacker obtained the KRBTGT password (the one that encrypts TGTs) or related hashes, they would be able to create as many tickets as they want, impersonating any user, and freely establishing policy parameters. Moreover, KRBTGT password is hardly ever renewed in most organizations (it could cause significant disruptions, due to legitimate tickets becoming invalid), providing the attacker with nearly full access to the domain for a long period of time.
  - **Silver ticket** A limited version of the golden ticket. It can be created by an attacker who is able to extract the password or hash of the account that encrypts TGSs. It would also provide unlimited access but only to a particular service (the one whose key or hash was leaked).
- **Pass-the-ticket** Windows does not allow extracting Kerberos tickets from a particular machine, but there are tools that perform this operation, providing the intruder has enough privileges [44]. If an attacker extracts a ticket, they can use it on another machine, being able to impersonate the owner until the ticket expires. If the ticket was a TGT, it would provide access to every service on the domain. On the other hand, service tickets only provide access to a particular service.

Kerberos proxiable tickets By default, tickets are granted to a node, as long as it is managed by the principal they claim to represent. However, when more flexibility is needed, proxiable tickets may be used. According to the Kerberos standard [39], when a client sends a ticket with the *PROXIABLE* flag enabled to a server, the server can send it to the TGS to request a new client-to-server ticket. The TGS will the return a ticket based on the received one that allows the server to access the service as if it were the client. These tickets must specify the network address from which they can be used (i.e. the server address), reducing the attackers' ability to stole credentials.

In Active Directory, Microsoft Windows directory service, proxiable tickets are included in a functionality called **Kerberos delegation** [45]. It provides means for authorized servers to be able to authenticate on behalf of their clients. It is not mandatory, but limiting the server's ability to impersonate users to partic-

ular services can be done on a per service user[4] basis.

[4]**service user**: User account that does not represent a user, but a service; providing a security context for the service execution.

3 Background

The background chapter extends the theoretical foundation by covering a set of definitions that are especially relevant for this thesis work. Particularly, it provides a generalised view of the characteristics of the target architecture and of the problem that this thesis is addressing.

3.1 Industrial Control Systems

Industrial Control System (ICS) is a term that includes a wide range of control systems, used for controlling industrial processes. They are found in most industry sectors (e.g. electrical, water, oil, chemical, transportation...) and they commonly control critical infrastructures [46].

They simplify process control by translating operators' commands into actions on physical components and/or by providing an interface for accessing data generated by those components.

Since ICSs support industrial processes in critical infrastructures, their protection is a priority. In fact, there are standards in several countries that define the security measures required by law for protecting ICSs [46].

Nevertheless, even when industrial processes controlled by ICS cannot be considered of critical nature, their protection is also important. Not only due to the economical consequences of the disruption in any industrial operation, but also because of their safety implications. As previously stated, commands to ICSs produce actions on physical components. Thus, their protection is not only a security matter, as with traditional computing. Attacks on ICSs can also produce safety hazards, including life-threatening situations.

ICSs rely on field controllers in order collect data and perform physical action on the controlled processes. Depending on how they interact with the physical components and their functionality on the whole ICS infrastructure, ICS controllers can be classified in three main categories [47]:

- **Programmable Logical Controllers (PLCs)** run programmed instructions depending on the readings from the sensors and the orders of the controllers, moving actuators or changing switch settings. They are designed to withstand difficult environmental conditions and can be deployed for long periods of time (more than 10 years).
- **Remote Terminal Units (RTUs)** are electronic devices controlled by a microprocessor. They are remotely managed, providing data to the control centre, which can issue commands to RTUs to be performed on the physical processes. Their differences from PLCs are difficult to define, since RTUs are implementing functionalities typically found in PLCs. However, it is common that RTUs are located in remote areas, far from the control centre and potentially using Wide Area Network (WAN) protocols. On the other hand, PLCs usually control processes in factories, where the control centre is in the same building or area and local networks can be used for communication.

- **Intelligent Electronic Devices (IEDs)** devices are and advanced variation of PLCs. Just as them, they can perform programmed control operations, based on the data from the sensors they manage. However, in contrast with PLCs, IEDs are able to control different aspects of each equipment piece, consequently offering superior control functionalities. Particularly, they provide protection, control, monitoring, metering and communication.

ICS supervisory systems can be classified depending on their functionalities and on the nature of the separation between them and the field components. A list of most common ICS types is included below [47]:

- **Building Automation Systems (BASs)** control and monitor the services present in a building (e.g. security, power, heating, fire protection...).
- **Safety Instrumented Systems (SISs)** monitor industrial processes to prevent safety hazards, by taking systems to a safe state should preestablished conditions are violated. [48]
- **Process Control Systems (PCSs)** control the automation of one manufacturing process at one site
- **Distributed Control Systems (DCSs)** control several automation processes at one site. They may monitor several PCSs, or the automation of a whole plant.
- **Supervisory Control And Data Acquisition (SCADA)** systems provide an operator in a remote location information about the state of an automation process and the ability to issue commands to modify the state of the that process [49]. They are similar to DCSs, but they are able to control remote processes that are potentially distributed across a wide geographical area.
  - **Energy Management System (EMS)** is an specific type of SCADA system designed for managing the generation and distribution of electricity in a national or even international scale.

3.2 SCADA constrains

When compared to traditional IT systems, SCADA systems have particular characteristics that must be considered for designing an architecture that includes them. They affect the likelihood of vulnerabilities being present on those systems and add computational limitations and potential compatibility issues that may affect any new deployment.

SCADA systems control real-time industrial process. Therefore, when an operator uses them, they should be able to see real-time data of the current state of the field components. The effect of delays in SCADA systems has been analysed by several researchers. For instance, J. Luque et al. proposed an analytic model for studying it [50].

Delayed data could lead to improper responses by the operator, based on information that is not longer valid (i.e. the current state of the system is different to the one they are shown). Similarly, the system would be negatively affected if user commands were delayed, even when sensor data arrives in real time to the operator.

Therefore, latency is an important performance measurement in SCADA systems. Maintaining a low latency can avoid delay-related errors, aiding in reaching a good service level in the whole SCADA-managed infrastructure.

In general, it is common that SCADA systems are deployed on old and out-of-date equipment. Upgrading is commonly faster in traditional IT systems than in industrial control systems [51]. Particularly, it is common that SCADA software runs on old devices, since replacing them would likely disrupt the processes under supervision. For the same reason, patches and updates are not applied as fast as in traditional systems.

Using legacy equipment and software can lead un-patched vulnerabilities and limited performance. As a result it is important to consider the possible presence of legacy equipment and its consequences when designing for SCADA-related infrastructures.

Moreover, in any microprocessor, software processes fight for computational resources, particularly CPU time. Since no system has endless resources, the more processes there are, the less CPU time each one has. Resource limitations have a potentially significant impact in industrial control systems.

If a process did not have timely access to the computational resources it needs, the service(s) it provides would be potentially delayed, affecting the latency of the system. Legacy systems are potentially more sensitive to CPU overuse, since they are typically more resource-constrained. Moreover, even when resources are enough, the use of CPU affects power consumption. Higher CPU usages cause higher microprocessor and heat-dissipation energy utilization, which ends in increased costs for the owner of the system.

As a result, due to the likely present of legacy systems, the criticality of industrial processes and the potential power-consumption cost impact, a design targeted to a SCADA architecture must consider the CPU usage as an important benchmark for evaluating its suitability.

Furthermore, many countries have regulations regarding SCADA systems and their security. Therefore, when applying any modification to SCADA infrastructures, the fulfilment of requirements established by ICS regulations must be kept.

Finally, in contrast with traditional IT systems, industrial control systems interact with the physical world. Thus, the consequences of attacks against SCADA systems could cause safety hazards, and even lead to life-threatening situations. In other words, SCADA systems protection is not only a matter of security, but also of safety.

3.3 SCADA security procedures

Industrial Control systems are typically more slowly updated than traditional IT systems [51]. One of the reasons behind this backwardness is the consideration of availability as a top prioirity for ICSs, since any disruption can have economical and even environmental or safety consequences. Moreover, ICS systems used to be isolated from external networks, creating a thoughtless overconfidence on the system resistance against attacks that minimized the perceived importance of security updates.

However, in order to improve automation processes, new functionalities need to be implemented. Organizations developed high-level control systems based on traditional IT, that where afterwards integrated with existing automation infrastructure. Thus is becoming more common that corporate and industrial networks are integrated, providing more functionalities, but increasing the attack surface and making the threat landscape of industrial information systems similar to the corporate systems one. Attackers do not ignore their new entry points, leading to an increasing trend in attacks against ICSs. [51][52].

As a result, is it important to follow recommendations issued by state institutions and in most cases mandatory to meet the security requirements they establish. In the US, the NIST and the American National Standards Institute (ANSI) are the regulatory bodies that are responsible for SCADA systems and their security [53]. On the other hand, in the EU, cyber security recommendations and regulations, including those related with Industrial Control Systems, are issued by the European Union Agency for Network and Information Security (ENISA) [54].

As a general rule, it is crucial that SCADA systems and any related developments are designed with security in mind, in order to offer the best protection against potential attacks. One important aspect of security is the accountability of commands. Any organization should be able to determine which user requested the execution of each action, in order to ensure that wrongful actions can be prosecuted and to manage incident response procedures.

Moreover, accountability, like any security property, must be as resilient as possible to system compromise. In other words, the number of nodes whose compromise would keep the system from providing accountability must be limited.

Accountability is only valid in environments where authentication can be trusted. The link between an user and their actions is completely dependant on them being authenticated. Otherwise, impersonation is possible and records cannot be trusted. Thus, non-repudiation would not be provided.

As a result, authentication must be considered a hard requirement. Every node, and particularly those that perform industrial control operations, must be able to authenticate the source of any command, even when it is not received directly, but forwarded by a third party. Source authentication in the presence of middle nodes is called multi-tier authentication, covered in the next section (§3.4).

3.4 Multi-tier authentication

As stated in §2.1, authentication is the process that ensures that an entity is who it claims to be. A simple authentication operation involves two parties: an entity A that sends a message and an entity B that receives it and verifies that entity A is who they claim to be.

However, there are situations where A and B do not communicate directly, but there is one or several middle nodes (M_i) that forward messages from A to B and may also need to authenticate the source. In that architecture, the authentication process is defined as multi-tier or multi-hop authentication. Figure 4 compares this operation with its simpler counterpart, single-hop authentication.

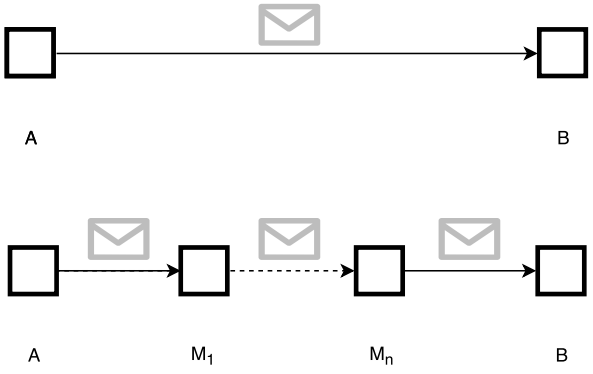

Figure 4: Single-hop and multi-hop authentication

The main challenge of multi-hop authentication is the need to provide means of verification of the source identity of a particular message at several nodes. The most simpler approach is to authenticate and trust each forwarding node to identify the initial sender. A source identity field would be a easy implementation of this solution. Kerberos also provides this functionality (Kerberos delegation), by the use of proxiable tickets, that allow a node to impersonate the original sender.

However, trusting the the forwarding node to verify the identity of the initial sender makes the system vulnerable to the compromise of any of the middle nodes. Should an attacker have access to any of them, they would be able to change commands, impersonating any user. Kerberos delegation, reduces the surface of the attacked by limiting the possible target users to those whose ticket in the server has not expired. Nevertheless, the vulnerability is still present.

Moreover, multi-hop authentication solutions based on symmetric cryptography need the sender to know the destination in order to obtain a key shared with them, reducing the operational flexibility at the middle nodes.

On the other hand, asymmetric cryptography provides a way to authenticate the sender regardless of the destination. As a result, it is a common solution for solving the multi-hop authentication problem. As long as a node can link a public key and an identity, any messages coming from that entity can be verified.

The main challenge with asymmetric-key based authentication is the need for

associating an identity and a public key. Typically, a Public Key Infrastructure (PKI) is established to provide this functionality by the use of certificates and the establishment of a Certification Authority.

4 Methodology

This Master thesis methodology relies on the design science paradigm. Most Information Systems research rely either on this paradigm or on behavioural science [55]. The former focuses on the creation of "innovative artefacts", while in behavioural science the purpose is to explain or predict human behaviour [55].

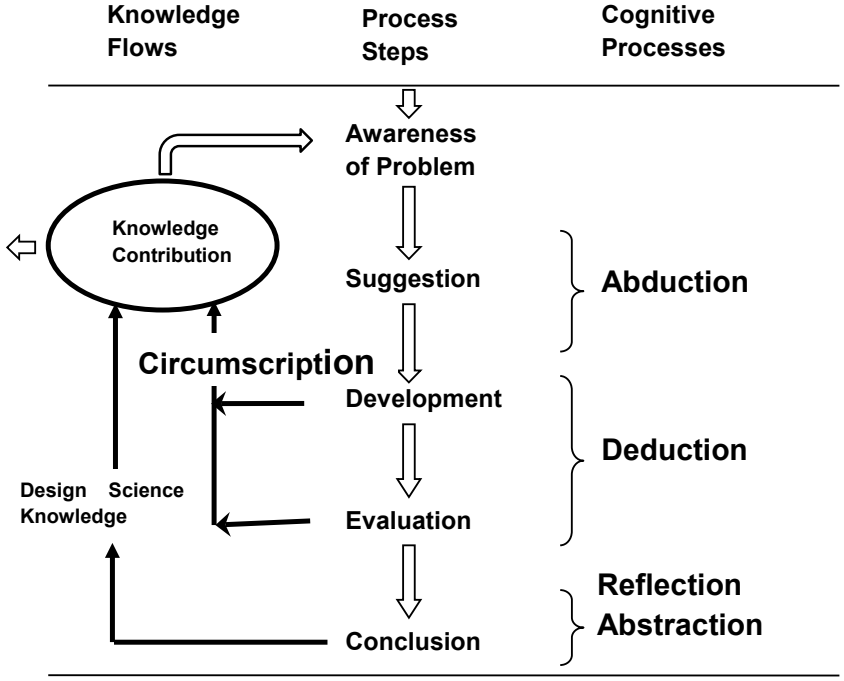

Figure 4. Cognition in the Design Science Research Cycle

Figure 5: Design Science Research Cycle [4]

Consistent with the design science research model previously described, research begins with *Awareness of a Problem*. Design science research is sometimes called "Improvement Research" and this designation emphasizes the problem-solving/performance-improving nature of the activity. **Awareness of Problem** The introduction chapter and the need for the research conducted in this thesis, defining the existing problem that we are addressing. This problem is specifically defined for our target environment in the requirements chapter.

Suggestions for a problem solution are abductively drawn from the existing knowledge/theory base for the problem area (Peirce, 1931). These suggestions may, however, be inadequate for the problem or suffer from significant knowledge gaps (which make the problem a research problem). Using existing knowledge, an attempt is made at creatively solving the problem. The solution—a tentative design—is used to implement an artifact in the next phase shown as *Development* in the diagram. Partially or fully successful implementations are then *evaluated* according to a functional specification (sometimes implicit) during the *Evaluation* stage. *Development*, *Evaluation*, and further *Suggestion* are frequently iteratively performed in the course of the research effort. The basis of the iteration, the flow from partial completion of the cycle back to *Awareness of the Problem*, is indicated by the *Circumscription* arrow. *Conclusion* indicates the end of a research cycle or the termination of a specific design science research project.

Suggestion The analysis chapter studies several potential solutions to the research problem and conducts a qualitative comparison of them. The investigative process started from high-level granularity options, namely asymmetric and symmetric cryptography. Then, they were extended by definite different alternatives within those two main categories.

Finally, one of the potential designs was selected, after comparing it with the other options in terms of security level, performance and implementation costs.

Knowledge contribution resulting from new knowledge production is indicated in Figure 4 by the arrows labeled: *Circumscription* and *Design Science Knowledge*. The *Circumscription* process is especially important to understanding design science research process because it generates *understanding that could only be gained from the specific act of construction*. Circumscription is

Thus, the result of the analysis chapter is a solution suggestion, whose explanation is extended in the design chapter. Therefore, both chapters represent the suggestion step in this thesis.

Moreover, even though the theory and the background chapters do not strictly belong to any of the steps we defined, they are the basis of the analysis chapter. They cover the research done before the analysis and the information they content is crucial not only for performing the analysis of possible solutions, but also for potential readers of the thesis to understand that analysis.

Development The implementation chapter can be identified with the development step. It details the prototype implementation of the proposed design. In other words, it describes the construction of the artefact. The purpose of this step is not to develop an state-of-the-art final product, but to demonstrate the validity of the suggestion.

The product of this stage not only includes the related chapter on the thesis, but also the code written for generating an actual working artefact. Following the design, a basic implementation was written, that performed the minimal required operations for considering a valid instance of our design. It consisted in the configuration of the AD server for storing public keys and on the generation of two console applications that exchanged messages between them: the signing and the verifying applications.

However, once that Minimum Viable Product (MVP) was produced and tested, new features were added progressively to address certain limitations when applying the design to the target architecture:

- A Graphical User Interface (GUI) was added to the initial console application in the signing node.
- The RSA algorithm was included as an option in the verifying node and as an extra standalone application in the signing node.
- A purge script was created for periodically removing obsolete keys from the AD.
- Two alternative solutions were proposed for avoiding high delays in the key exchange process due to the low frequency replication of AD domain controllers when they are logically placed in different sites.

Therefore, our implementation relies on a prototyping approach to construct the final artefact. Particularly, our coding process is based on experimental prototyping [56]: we focus on the technical implementation of our design.

Moreover, our final artefact must not be considered as ready for deployment, since, as stated before its main purpose is to demonstrate de validity of the suggestion. A product ready for production environment falls out of the scope of this thesis, as it would require to adapt it to different ABB proprietary protocols and to test it in more realistic simulations.

Evaluation Both the performance evaluation and the discussion chapters can be identified with the evaluation step. The discussion chapter assess to which

extent requirements are fulfilled (i.e. how well the problem is solved). This assessment is based on the performance evaluation for parameters that could be measured in our mock implementation. It consists in a quantitative study of the effect of our solution on the target architecture. However, the fulfilment of other requirements, such as the required level of security, is intrinsically qualitative and thus more difficult to assess. The discussion section relies on the theoretical foundation and on the analysis to decide to which extent our work fulfils those requirements.

Conclusion The conclusion step is covered in this thesis in the chapter with the same name. It summarizes the process, the results of the thesis and whether the research questions are answered. Furthermore, it suggests how our research can be extended, by providing several future work possibilities.

Communication Some authors explicitly consider an additional final step that lies in communicating the result to the appropriate audiences [57]. This whole thesis document and its presentation in KTH represent the communication step.

5 Requirements

This chapter establishes the requirements that the solution to be proposed by this thesis must fulfil. Establishing a set of requirements not only provides guidance for delivering a design and its implementation, but also helps validating them.

5.1 Architecture constrains

Studying the target architecture is crucial. Not only for determining the costs and the impact of implementing the proposed solution, but also for maximizing compatibility with the existent deployment. Current architecture is shown in Figure 6. Operators connect to the SCADA servers through an access server, that acts as a proxy for their commands.

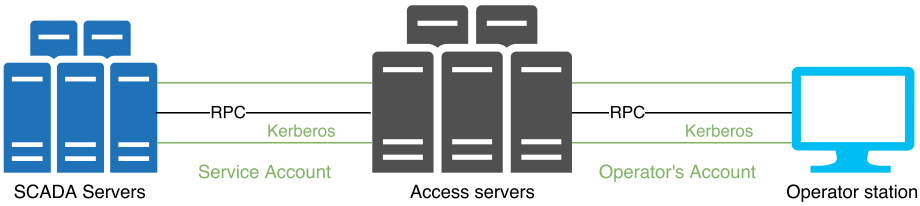

Figure 6: Simplified representation of the current architecture

The connection between the user station application and the access server is based on Remote Procedure Calls (RPCs)[5]. On the target architecture, RPC messages are sent over a Kerberos tunnel, thus confidentiality and integrity of the messages sent to the access server is guaranteed. Moreover, authentication of the operators is also ensured, since users are required to log in with their Kerberos credentials to be able to send commands to the access servers.

Once a message arrives to an access server, it is forwarded to the appropriate SCADA server. As for the previous step, the communication is RPC based and protected by a Kerberos tunnel. Nevertheless, the access server is authenticated against the SCADA server using a service account. The same one is used regardless of the operator that made the request. Thus, accountability cannot be ensured, since the SCADA server does not know which user issued each command.

The proposed design needs to consider this architecture. Particularly, the access server needs to be kept in the middle of the connection between the operator station and the SCADA servers. Its presence provides scalability, by forwarding commands from different sources using an unique connection to the target SCADA server.

The solution needs to focus on ensuring accountability at the SCADA servers, while also providing authorization at the access servers. In other words, access servers should only forward commands sent by authorised users and SCADA servers must know who is the issuer of each command.

[5]**RPC**: Inter-process communation technology that provides location transparency. Calls to remote procedures can be done as if they where local.

Finally, the new design should be as transparent to the user as possible. Particularly, users should not need to log in or provide their credentials more than once.

5.2 Security requirements

As previously stated, security in SCADA systems is a priority, and enforced by several regulations in some countries. Our design should be secure enough to comply with these requirements.

The main requirement of our design is that accountability is provided at the SCADA server, while the access server continues to manage the authorization for issuing commands. The desing should be resilient to the compromise of the access server. Even if an attacker takes partial or total control of an access server, they should not be able to send commands to the SCADA server spoofing the identity of an operator. Moreover, providing the access server is not compromised, it must not forward commands to the SCADA server if they are not issued by authorized users.

In addition, the current architecture already provides confidentiality and integrity at each step of the communication by using Kerberos tunnels. The design provided by this thesis must guarantee at least the same level of data protection as what is currently enforced. Nevertheless, improvements can be made. For instance, integrity may be ensured end-to-end (from operator station to access server).

Furthermore, protection against common attacks against security protocols must be in place:

- **Man-in-the-middle (MITM) attacks** They consist in relaying the communication between to parties, acting as a illegitimate proxy. The total avoidance of this attacks is out of the scope of this thesis, since it involves network administration policies. However, by encrypting and verifying the integrity of the communications their effects can be mitigated.
- **Replay attacks** They are based on sending valid transmissions repeated or delayed. An attacker could re-send commands issued by an authorized user or even manipulate the network to produce a delay on valid commands. As a result, commands should have an expiration time, while leaving a margin for message Round Trip Time (RTT) (currently, messages expire after 25 seconds). Moreover, commands must be considered only once. Subsequent repetitions of the same command request should be ignored and possibly logged.

5.3 Performance requirements

Broadly, there is a trade-off between the level of security that a particular architecture offers and the communication latency and CPU usage of the involved systems. Cryptographic operations have computational costs, especially when using public-key algorithms. Moreover, they usually imply overheads on the

message size or count (i.e. messages to a third-party authentication server or protocol overheads). Compression can help mitigate this overhead, but it also generates computational costs.

SCADA operations are potentially critical. Thus, the time frame between the issuing of a command by the operator and its execution by the SCADA server must be as short as possible. Current ABB deployments manage to keep this latency under 1 second. The proposed design must not suppose a significant increment on the latency of command execution.

Finally, computational resources must also be preserved. If the new architecture needs to perform more intensive operations, a greater number of CPU cycles would be used. This need for more CPU cycles could produce increments in energy costs. It could even lead to current systems not being able to run the new architecture. Therefore, the proposed solution should keep computational resource utilization to similar or lower levels than the current ones.

6 Analysis

Building up in the background and the theoretical foundation, this chapter will discuss the possible designs that would fulfil the previously established requirements. Namely, three possible solutions and several variations are considered: One is based on Kerberos delegation, while the other two maintain the current Kerberos architecture, and provide authentication either by using public-key or symmetric-key cryptography. Digital signatures are considered as a an example of public-key authentication and symmetric-key solutions are evaluated by analysing the particular case of using HMAC with Kerberos session keys.

6.1 Trusting the access server: Kerberos delegation

Kerberos delegation is the most straight-forward approach for fulfilling the requirements, since the current deployment is already Kerberos-based. Therefore, this solution would only need to give the adequate permissions to the access servers so they can authenticate against the SCADA server as if they were the client.

This alternative would be completely transparent to the user, not even needing to modify client-side applications. Changes would be needed in the Active Directory Domain Controller, enabling Kerberos Delegation for the access servers. Afterwards, access servers would need to be set up so they use the client credentials for authenticating against the SCADA servers. Finally, accountability of user commands should be enabled at the SCADA servers, ensuring that each command and its issuer is recorded, and non-repudiation provided.

Performance The main advantage of Kerberos delegation for the target architecture is its low latency and CPU usage impact that it would have. At the client side there is no impact, since no modifications are needed in that tier.

Moreover, SCADA servers already record the issued commands. Thus, the only overhead at this tier consists in reading the user name during the client authentication process. This operation is computationally simple and should not significantly affect latency or CPU usage.

In addition, at the access servers, the difference from the current design is that instead of using a service account ticket for authenticating against the SCADA server, a client to server ticket would be in place, requested using the received proxiable ticket. It is likely that more requests to the Authentication Server (AS) are needed, since one ticket would be requested per user, but those tickets will be cached by the access server and valid until expired. Therefore, only commands from new users or those sent just after the client ticket has expired are likely to have a higher latency.

In any case, symmetric key operations are not very computational expensive and the ASs are typically installed on the same network as the servers. Thus, neither latency nor CPU usage would be substantially affected at the access servers.

Security The main limitation of this design is the level of trust that it gives to the access servers. Their power impersonating the users should be contained, by limiting their delegation capabilities only to the services provided by the SCADA services.

Nevertheless, even if limitations are imposed, access servers would be still be very powerful. They would be able to send commands using the credentials of any user that has recently logged in to the server. Thus, if the access server is compromised, the trust on the accountability at the SCADA servers is broken, and non-repudiation not provided.

Conclusion In summary, Kerberos delegation would be easy to implement over the current architecture and would not significantly increment latency or CPU usage. However, it relies too much on the access servers. The compromise of an access server would lead to a lost of trust on the authentication at the SCADA servers, since the server can send commands impersonating any recently connected user. Consequently, it would not be possible to properly authenticate commands and non-repudiation would not be provided.

6.2 Symmetric-key cryptography: HMAC over the current Kerberos architecture

Currently, users authenticate against the access servers using their user accounts. When the access server forwards messages to a SCADA server, it is authenticated using the server service account. Since Kerberos relies on symmetric-key cryptography, solutions that are also based on this type of cryptography are potentially simpler to implement, since key management can be done by Kerberos. There are several methods of providing integrity and source authentication relying on symmetric-key, being keyed-Hash Message Authentication Code (HMAC) one of the most common.

In this set up, the client would need to request two client-to-server tickets and corresponding session keys. One would be meant for authenticating against the access server, by presenting it during Kerberos protocol authentication. The other ticket, designed for authentication of the client against the SCADA server, would be sent as part of the message. Particularly, the message would include the command, the mentioned ticket, a timestamp value and a nonce value. Together with the session key associated with the ticket, it would be the input for generating the HMAC, which would be appended to the message when sending it. Figure 7 shows the message construction process.

At the access server, there is no significant difference from the original behaviour. It would read the command from the message if needed, while the rest of the content would be ignored, since authentication of the client is provided by the Kerberos protocol. When forwarding the message to the SCADA server, an immutable part should be kept as received from the operator, including all parameters and the HMAC.

Finally, the SCADA server would receive the message with the appended HMAC. The access server is authenticated via the Kerberos protocol. For authenticating the operator, the SCADA server would need to extract the session key from the

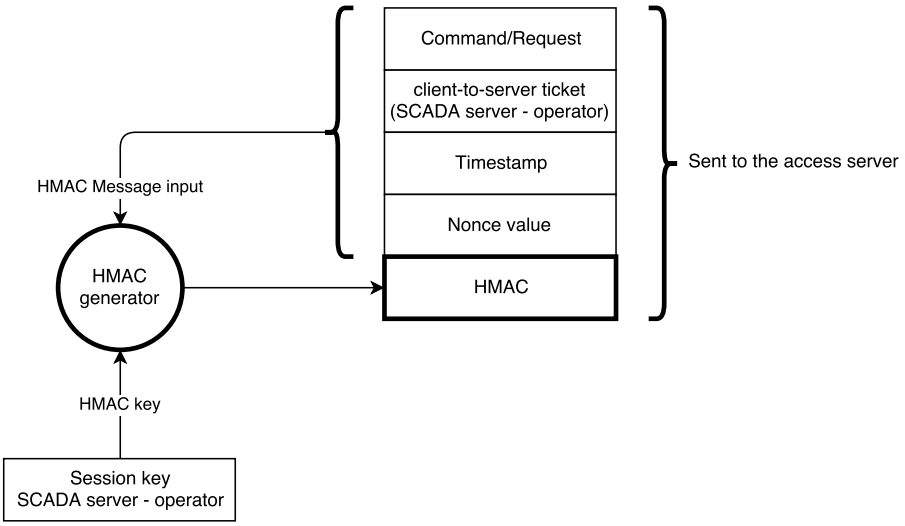

Figure 7: Message with HMAC sent to the access server

ticket on the message, that also includes the operator's identity information. Using that session key and the message, an HMAC can be generated from the immutable part of the message and compared to the appended one. If there is a match, the operator is authenticated and the command can be executed and recorded for accountability.

Performance The process performed at the operator's station implies requesting an extra ticket if there is no valid cached ticket available for authenticating the operator against the SCADA server. Therefore, latency would be affected when issuing commands for the first time or after ticket expiration. Moreover, HMAC generation computational cost depends on the hashing algorithm underneath. Even when using secure algorithms, such as SHA-256, CPU utilization is low [58], particularly for inputs as short as the ones expected for this architecture (text based commands). Thus, the main overhead at the client would be the ticket request operation, which would not be needed for every command.

In addition, there is no performance or latency overhead at the access server, whose functioning is the same as before.

Furthermore, the overhead at the SCADA server is low. It needs to decrypt the ticket, using its key with the KDC. Since it is a symmetric key operation, it is not computationally complex. Moreover, the HMAC has to be generated for comparing it to the one appended to the message. As stated before, generating a hash does not produce an important computational overhead.

Security When using Kerberos, the trust lies in the KDC. Similarly, since this solution relies on Kerberos for key management, the KDC is also the source of trust. It generates the session key to be used in the communication by any two parties. Thus, if the KDC is compromised, session keys would be exposed,

and users would be potentially impersonated.

However, in comparison with Kerberos delegation, the HMAC solution offers better authentication and integrity protection, meeting those requirements even if an access server is compromised. By maintaining the current architecture, confidentiality, integrity and node authentication in node-to-node connections (e.g. operator to access server or access server to SCADA server) are provided.

On top of that, HMAC offers end-to-end integrity and source authentication, by guaranteeing that the issuer of any message is in possession of the session key used for generating the HMAC. That session key should only be known by the KDC, the sender and the receiver. Therefore, as long as the receiver trusts the KDC, the message HMAC is enough proof that the message was originated by the user with whom it shares the session key and who is identified by the information contained in client-to-server ticket.

Conclusion This solution does not incur in great installation or management costs, since the current architecture is kept. Moreover, the solution highly relies on the already available Kerberos infrastructure, only requiring small modifications at the SCADA server and in operator applications to implement HMAC, and on the access servers to keep an immutable HMAC'ed part when forwarding messages. In addition, CPU usage increment is low, since hash operations are computationally efficient. Latency would only be affected in some commands, due to ticket requests.

Nevertheless, the HMAC solution has an important drawback. Since messages are signed using a session key tailored to a particular service, the user would need to know beforehand which service account is running in the destination server. Thus, there is no flexibility in the access server for forwarding commands, reducing its utility.

6.2.1 Unknown server HMAC-based authentication end-to-end

The lack of flexibility of the HMAC-based solution could be addressed by implementing a method for requesting client authentication as a response, similar to HTTP 401 messages. If an end server receives a message from an access server, that does not include an HMAC or whose HMAC was not generated with a session key established by the KDC for that server and client, it answers back with an "authentication error" message, stating its identity.

When the client receives that response, forwarded by the access server, it would request a ticket for the server, according to the identity information in the response. Then it would send the command again, but this time with the HMAC generated with the session key for client and end-server.

This extension for HMAC based authentication, fixes the flexibility problem of this solution. Moreover, it is still stateless, not needing any extra infrastructure, like public-key solutions commonly do. However, should there be authentication errors at the SCADA server, latency would be significantly incremented. The message would need to be sent twice (even more if there is more than one middle server) and extra ticket request(s) would be required. As a result, latency would

be at least twice as usually, reaching unacceptable levels.

6.2.2 Unknown server HMAC-based authentication with access server implication

The previous solution main drawback is its latency. However, if the access server takes part in the HMAC-based authentication process latency can be reduced. The sender could include who the HMAC is destined to (if any) in the message. Thus, the access server would be able send and authentication error if the HMAC for the final destination is missing, avoiding forwarding the message to the final server if it cannot be authenticated at that node.

Nevertheless, the access servers would need to check a new field on the messages and they must be able to send "authentication error" messages, leading to more significant implementation effort at the access servers. Moreover, messages would still need to be sent twice. Even if the first message is not forwarded to the final destination, latency is significantly incremented when compared to the current architecture.

6.2.3 HMAC-based authentication for a pool of servers

When using HMAC there is a trade-off between flexibility and latency. If the HMAC is generated for a particular server, only that server can rely on it for authenticating the sender of the message.

On the other hand, if HMACs are generated on demand, latency is increased, as messages would need to be sent twice. The first communication is needed for the access server to decide which server is the final destination and reply back to the issuer with an authentication error message. The second delivery is actual message sending, with the HMAC for the destination server attached.

A middle-ground solution is also possible. It would be based on a pool of possible final destination servers. A list of them would be available at the access servers, so users can update it periodically. In order to send a message they add the HMAC for every server on the list. Therefore, any of the destination servers would be able to corroborate the sender's identity, by verifying the HMAC generated with the key shared between that server and the sender.

When sending a message, a user would need to request a ticket for each possible final destination server, and append an HMAC for each of them. Thus, this solution is only suitable if the pool of servers has a small size. With a high count of possible final servers, the number of ticket requests would notably affect latency and message size would be significantly increased, due to the number of appended HMACs.

6.3 Public-key cryptography: Digital signatures over the current Kerberos architecture

6.3.1 Public Key Infrastructure for credential management

As stated in §2.5.1, digital signatures are a common technology for user authentication. They use public-key cryptography for securely validating identities. In order to manage user credentials, it is common to establish a Public Key Infrastructure (PKI). Particularly, a Certification Authority (CA) would be set up and certificates for every user and server issued. Moreover, revocation capabilities should be included for maintaining security after credential theft or loss.

This solution would rely on sending the commands together with a digital signature of a part that remains unchanged end-to-end. Messages, and thus the signature input, must include a nonce value, to avoid them being reissued by an unauthorized party, and a time stamp, so delayed messages are not considered. The digital signature would be generated using the operator's private key. Thus, any receiver can verify the identity of the operator, provided they trust the CA that issued the operator's certificate.

Using digital signatures passes some responsibilities to the user, since they need to generate their key pairs, get their credentials (digital certificate) verified by the CA and follow best practices for protecting them. Security could be improved by using cryptographic cards, designed to store PKI certificates. They would reduce the risk of compromise, as they have better protection mechanisms than software-based PKI credentials storage, but they also imply higher costs and a greater burden to users, who would be provided with an extra physical device.

Furthermore, client-side applications would also need modifications to be compatible with this design. Particularly they would need to have access to the client's certificate and use it to sign every issued command.

Nevertheless, access servers would not need any modification, since they can keep using Kerberos for authenticating the client and the SCADA servers. They just need to preserve the immutable part of the message and its signature present on the received commands when forwarding them.

In addition, SCADA servers would need to verify command signature before executing and recording them. For that purpose, they must have the certificate of the trusted CA that signed client certificates. When issuing the commands, the time stamp and the nonce must be checked against the database and the system clock, to avoid replay or delay attacks. It would also discard legitimate commands delayed for technical reasons, avoiding unexpected consequences in the system. After commands are executed, in order to provide non-repudiation and to be able to verify subsequent ones, they should be recorded into the database, together with the user name of the operator, the time stamp and the nonce.

Moreover, although it is not a requirement, authentication of SCADA server responses can be also implemented, by signing the reponses in the SCADA server, using a certificate issued to it. The client application would then verify

those responses by validating their signature.

Performance This design would have greater effects in CPU usage than Kerberos based solutions. As stated in §2.3, public-key operations computational cost is high, in comparison with symmetric key operations. Thus, CPU usage would be affected, both at the client node, that needs to sign the commands before sending them, and at the SCADA server that needs to validate the signature before executing each received command.

Furthermore, the degree to which latency would be affected by cryptographic operations would depend on the computational power of the SCADA server and the operator station. However, in contrast with Kerberos delegation, tickets would not be used for authentication of the client at the SCADA server. Thus the latency increment related with those ticket requests would be removed. As a result, the latency increment produced by cryptographic operations can be compensated by the reduction in ticket-related communications, as long as signing nodes have enough computational power.

Moreover, certificate revocation, when done using CRLs, would only affect latency and CPU usage when the list is being updated. What is more, certificate generation operations have a high computational cost, and thus CPU usage, particularly when using RSA. However, certificates are commonly valid for long periods of time (e.g. months or years), making this CPU usage peak rare and, consequently, not very significant.

In addition, certificate revocation would also affect the CA, that needs to keep track of revoked certificates, and any node that performs signature verification. Nodes would need to periodically download CRLs or to perform requests using the OCSP protocol for each verification operation. For this architecture, the former is a better option, since it is unlikely that it would affect the latency of the verification operations as it can be done regularly (e.g. daily), when system load is expected to be low.

Security Public-key cryptography, and consequently digital signatures, offer a very high level of security, by uniquely identifying a user. In contrast with symmetric-key based solutions, including HMAC or Kerberos, digital signatures can be used regardless of the receiver of the signed payload. In other words, a signed message can be verified by any receiver, providing they trust the CA the issued the certificate of the signing party. Moreover, the receiver does not need to know the private key. Thus, signing a message proves that its issuer is in possession of the private key associated to the signature, but does not require the receiver to know it.

In addition, public-key cryptography increases the security of the credentials sent over the network. In contrast with symmetric-key solutions, neither the private keys nor their hashes are sent over the network, reducing the success potential of brute-force attacks. They would only be possible against public keys, designed to withstand them. Moreover, if cryptographic cards were used, private keys would never be exposed, as long as the card is in possession of the legitimate owner and its security is not broken (i.e. card vulnerabilities).

However, if PKI is used for credential management, public-key cryptography security lies on the CA. The same way as with the KDC when relying on symmetric-key cryptography, if the CA is compromised, the whole system is exposed too, since the intruder would be able to generate certificates for any user and therefore generate valid signed commands. As a result, using digital signatures creates a new weak point to be targeted by malicious parties: the CA. It is an advantage for security, since it distributes security among different nodes and technologies (defence in depth), but it also increases the security team effort, by creating a new critical asset to protect.

Conclusion This solution provides a higher level of security than symmetric-key based ones, since, by using digital signatures, it unequivocally identifies the issuer of a command, even considering access servers to be compromised. Moreover, users can send signed commands without knowing which particular server is the receiver.

However, it has a high overhead in terms of installation and maintenance. It requires a PKI, and thus a server acting as a CA. Certificates would need to be issued periodically and after revocation, affecting the CA server itself and the clients that need to update them on their devices.

6.3.2 Active Directory for credential management

Due to the cost and effort of deploying a PKI, a solution that relies on digital signatures, while not needing extra infrastructure, has been studied. Should a PKI not be in place, an alternative method for the servers to corroborate user identities must be implemented. Particularly, users' public keys must be available to the server that is verifying their signatures.

Moreover, the target architecture uses Microsoft's Active Directory (AD) for managing user credentials. Therefore, for the purpose of reducing deployment effort and cost, this design relies on this service for storing users' public keys. Thus, users would not need to have their certificates signed by a CA, as no certificates would be required. On the contrary, they would only need their public key to be stored in the AD, associated with their user name.

Since it is also based in digital signatures, the overall functioning of this solution is similar to the PKI-based architecture. The user needs to append a signature to the messages it sends, so the receiving node can verify the identity of the sender, even it is not send directly (i.e. sent through the access server).

However, this functioning alone would not be enough for providing authentication end to end, since the absence of a PKI would make verifying the identity of the command issuer impossible. Thus, as stated previously, clients need to generate their key pairs and send them to the AD server. This server should support messages from authenticated users that want to add or update their public keys on the directory.

Asymmetric key pairs do not need to be generated for each message. Key generation is commonly done performed periodically (e.g. yearly) and the same private key used for signing several commands. Nevertheless, keys can be gen-

erated for each session, in a similar way to Kerberos tickets. While it would increment latency at the time when operators log in, it provides several advantages. Particularly, it would reduce key management effort, increase security on shared workstations and facilitate the use of several workstations by the same operator.

Moreover, in order to reduce the latency of message signature verification, receiving servers can cache users' public keys. Thus, public keys are only requested when the sender's key is unknown, and store locally on the server. Unlike when using PKI, since keys are generated for each session, there is no need for key revocation procedures, as keys would be dynamic and expire after short periods of time, like Kerberos tickets do.

In addition, for each message, the receiving end must be able to access the AD to consult the public key of the alleged sender and corroborate their identity, by verifying the message's signature. Since the AD server provides provides key management to the whole infrastructure, public keys stored there are available for every domain host, as long as the appropriate permissions are set.

Performance The computational efficiency of digital signatures is similar, regardless of the credential management technique. Signing and verifying of signatures and key generation would use the same amount of computational power when using AD than when using a PKI, depending on the length of the keys.

However, unlike with the PKI-based solution, there is no need for having a CA, eliminating the potential impact of running that service. Credential management is part of the AD server, that stores the public keys of the users and provides access to them. Since AD is already present in the architecture, providing access to users' credentials would not have significant performance impact.

Nevertheless, depending on how often users log in, latency could be affected by this solution. In general, when operators log in, keys should be generated and the public one pushed to the AD. Key generation is computationally intensive, particularly when using RSA. Moreover, receiving servers can use cached keys for short periods of time and update them after expiration or when a message is signed with a different key (potential workstation switch).

Moreover, if several AD servers exist, inconsistencies would be shortly present after public keys are pushed. Therefore, operators would not be able to use the system immediately after log in. Instead, they would need to wait until their public keys are propagated across the AD servers, so they are available to the receiving end.

Security As discussed before, digital signatures provide a great level of security, by ensuring unequivocal authentication, relying on asymmetric key cryptography. They identify the sender regardless of the receiver. Thus, unlike with symmetric-key based solutions, the destination identity can be unknown when sending the message. Moreover, the receiver cannot impersonate the sender, since the private key used for signing is only known by the sender.

In comparison with PKI-based digital signatures, this solution removes the need

of establishing a PKI, while providing a similar level of security. The main difference is that instead of having the CA as a source of trust, the AD server is the node that provides that service.

Therefore, the AD server is a SPOF[6], since it manages user credentials both for the Kerberos protocol and for the digital signatures. As a result, management is less complex, but failure or compromise of that server would have a great impact on the whole architecture. Redundancy and protection measures should be applied, but fall out of the scope of this thesis.

Conclusion As stated previously, digital signatures provide a high level of security, since they unequivocally prove that the owner of a key-pair is the sender of a message. However, using asymmetric cryptography usually implies the deployment of a PKI, incurring in extra management and implementation costs.

This solution is based on using Active Directory for storing users' public keys, instead of relying on certificates signed by a CA. Thus, the need for extra infrastructure is eliminated, while a equivalent level of security provided. Latency can be increased when operators log in, especially in environments with several AD servers, but it is not expected to affect the functioning of the architecture once public keys are distributed across all AD servers.

6.4 Discussion

In order to decide which method is the most secure and which one is recommended for implementation at ABB, several aspects must be compared. Namely, performance, security and implementation and management effort. This section will compare the possible solutions according to those parameters. Table 5 summarizes the results of that comparison from a technological perspective. For a deployment and implementation summary, refer to Table 6.

Latency In order to decide the performance impact of each solution, their latency and CPU usage effect must be evaluated. In terms of latency, every solution would generate increments in the time between command issue by the user and its execution at the SCADA server. This could be motivated by additional Kerberos ticket requests or due to cryptographic operations, such as HMAC generation or digital signing.

Some solutions need to send periodical messages. Digital signatures need updated public keys, which, when using PKI, is achieved with periodical synchronization. Similarly, when using the HMAC solution that targets a pool of servers, the server list nets to be up to date. This can be achieved using periodical synchronization or, for very static environments, by manual configuration in operator stations.

In any case, these messages do not need to be sent very frequently, since public keys are designed for long term usage and pools of servers are expected to be highly static. Therefore, even if they can affect latency of messages sent at the

[6]SPOF: Single Point Of Failure

same time, there are simple network operations that do not need a high amount of resources. Moreover, this synchronization operations should be performed in server off-peak times, to have potentially more available resources and avoid unwanted effects on critical operations.

Furthermore, Kerberos delegation effect on latency is minimal. When requesting a ticket for communicating with the SCADA server, the SCADA server does not use a service account. Instead, it asks for a ticket using the proxiable ticket received from the client. Since tickets are different for each client, even if they are cached, more tickets requests are to be made (one for each client). However, only with a high number of different operators sending commands this modification would affect latency. Even if that is the case, the impact would not be high, since ticket requests are simple operations and they are not needed for each message, only after ticket expiration.

Moreover, digital signatures use the current underlying architecture (service account authentication server-to-server). Nevertheless, when using AD for credential management, public keys must be checked after expiration, which can affect the latency of some messages. The main latency impact of this solution occurs at the time when the user logs in. The effect is not expected to be significant, since a lookup on the AD is a simple operation. However, operators' public keys need to be stored and, if many AD servers are in place, propagated so it can be available to the receiving end. This propagation operation could take several minutes, creating a delay at logging in time.

What is more, HMAC solutions need to request a ticket that contains the shared key between them if the server. When using HMAC for a pool of servers the amount of requests depends on the number of servers in the pool. Therefore, the higher the count of servers was, the greater impact in latency the HMAC would have. If tickets are cached, they do not need to be requested until expiration. However, the number of servers would affect the amount of needed HMAC calculations and the length of the message, having a latency impact even when no ticket requests are performed.

Furthermore, HMAC solutions that do not know the target beforehand have a significant latency impact. If end servers are the nodes that request the messages to have HMAC, latency is at least doubled, as messages must be send twice to the end server. On the other hand, if the HMAC is requested by the access server, depending on which node the message needs to be sent to, the latency impact is not as high, but still very significant. Messages are only forwarded once to the end-server but need to be sent twice to the access server.

Finally, both HMAC and digital signature operations can have an impact on latency. However, messages are not expected to be very long. Even with payloads of tens of MB, modest processors can perform digital signature and HMAC operations in short periods of time (in the order of milliseconds) [7].

CPU usage CPU usage impact would not be high in symmetric-key based solutions. This type of cryptography is not computationally intensive. Moreover, supporting operations for symmetric-key solutions, such as extra network requests (Kerberos tickets) or HMAC generation, are not CPU intensive either. On the other hand, digital signatures are based on public-key cryptography. As

Table 5: Comparison of the impact of the proposed solutions from a technological perspective. Each value is not absolute, but given in relation to the rest of the solutions

Solution	Latency		CPU usage	Security improvement	Periodical messages
	Message	Log in			
Kerberos delegation	Minimal	No	Low	Low	No
HMAC: One server	Low	No	Low	Medium	No
HMAC: Pool of servers	Medium	No	Medium	Medium	Yes
HMAC: Requested by access server	Medium	No	Low	Medium	No
HMAC: Requested by end server	High	No	Low	Medium	No
Digital signatures: PKI	Low	No	Medium	High	Yes
Digital signatures: AD	Low	Yes	Medium	High	No

a result, digital signing operations would generate a greater increment in CPU usage, due to the computational cost of public-key cryptographic operations.

The importance of the CPU usage increment is subject to the computational resources available at the server and at the operator's workstation. Benchmarks show that although public key operations are more resource intensive than their symmetric counterparts, they do not significantly affect CPU usage.

According to ENCRYPT II benchmarks, modern processors need less cycles for signing messages. But even if we consider a modest machine from 2010, with one 1700 MHz processor, results are acceptable. When using RSA 2048, signature verification of 59 bytes needed 86000 cycles, which is approximately a 5% CPU usage in 1 millisecond. RSA signing is more resource intensive than verification ($\sim$ 100 times higher cycle count) [7]. However, this operation is done at the operator's station, where CPU usage level is not so critical.

If digital signatures rely on Active Directory for credential management, key generation is performed more frequently than with PKI. Particularly, it is done each time an operator logs in. If using RSA, the number of cycles needed can reach significant values, with a high variance. ECDSA would solve that limitation, since it provides lower key generation times with consistent results across executions.

Security Every considered option provides operator authentication at the SCADA server. Thus non-repudiation requirement would be fulfilled regardless of the solution. However, the level of security provided by each solution is not the same.

Only Kerberos delegation authentication at the SCADA server is dependant on the access server, meaning that a compromised access server would break authentication at the SCADA server and, consequently, non-repudiation. The rest of the alternatives, provide authentication and accountability at the SCADA server, even after access exposure.

Moreover, digital signatures are more trustworthy than every other solution. Since they are based on public-key cryptography, only those who are in possession of the private key can sign messages. Thus, even if the SCADA server itself

Table 6: Comparison of proposed solutions depending from a deployment cost perspective. Each value is not absolute, but given in relation to the rest of the solutions

Solution	Extra infrastructure	Management effort	Implementation effort
Kerberos delegation	No	Low	Low
HMAC: One server	No	Low	Medium
HMAC: Pool of servers	No	Medium	High
HMAC: Requested by access server	No	Low	High
HMAC: Requested by end server	No	Low	High
Digital signatures: PKI	Yes	High	High
Digital signatures: AD	No	Low	Medium

is compromised, signed messages still provide non-repudiation. On the other hand, trust in symmetric-key solutions, is build upon shared-keys. Thus non-repudiation is provided as long as the recipient (the SCADA server) is trusted.

Furthermore, any of the solutions are vulnerable to the compromise of the credentials management severs. Symmetric-key based solutions rely on the Kerberos protocol for credential management, and are consequently vulnerable to KDC compromise. Similarly, public-key solutions whose credential management relies on AD would also be affected by the compromise of the KDC, which in Microsoft environments is typically hosted on the AD server. Finally, if digital signatures build over a PKI, trust would be broken should an attacker get access to the CA.

Implementation and management The implementation and management effort of each solution must also be considered for deciding which option is more suitable. Kerberos delegation would be the most simple solution to implement, since it would only imply giving the appropriate permissions to the access server and setting accountability at the SCADA server, based on Kerberos authentication.

The rest of the solutions have more complex implementations. In general, all of them need an immutable part of the message to be kept end-to-end, so the source can be authenticated based on the HMAC or digital signature of that string. In addition, there is a need to modify client and SCADA server applications to support HMAC or digital signatures, and the SCADA server must be able to read user identities from messages and verify them to provide accountability.

Moreover, while the previously mentioned implementation changes would be enough for one-server HMAC, the rest of the HMAC-based alternatives need even more modifications.

For instance, when using a pool of servers as target, there is a need for the operator to identify the HMAC for each server, in order to avoid unnecessary HMAC verification operations. Furthermore, with requested-HMAC alternatives, an implementation of authentication error messages needs to be consider, rendering these alternatives more complex.

Conversely, digital signature implementation is different depending on the credential management technique. If Active Directory is used, changes would be needed on the AD server to implement the public key field and on the communication end-points (operator and end-server) in order to store, look up and use the public keys.

Finally, when digital signatures rely on a PKI, implementation costs and required effort are increased. They are the only alternative that requires extra infrastructure, a PKI. Particularly, they need a CA that generates user and server credentials and whose certificate allows for verifying user identities. The CA itself would need a server to be deployed at, and would create management challenges. Specifically, it would be an additional critical asset to protect and would need access to operators' and servers' identities. Furthermore, certificate management must also be considered, since it would imply users and administrators, who would need to act after certificate expiration or theft.

6.5 Conclusion

In terms of security, digital signatures are the best option. With proper management, they provide unequivocal authentication, and better non-repudiation than symmetric-key based solutions. Moreover they meet latency requirements and their CPU usage impact should not be significant in modern processors.

Nevertheless, the target architecture does not have a PKI infrastructure in place, needed for digital signature management and certificate provisioning. ABB considers that the cost and effort needed for implementing and mataining a PKI infrastructure is not acceptable, making digital-signature based solution impractical if based on PKI credential management.

However, the need of PKI can be avoided by the use of Active Directory for credential management. It provides a security level similar to PKI-based digital signatures, while not having extra infrastructure needs. Its main limitation is the latency when the operator logs in. This latency impact is particularly significant when several AD servers are in place, due to the synchronization delay.

On the other hand, symmetric-key based solutions, are not perfect, since keys are shared between at least two parties. However, authentication can still be provided, understanding authentication as the possibility of the receiver to identify the issuer of a command. Nevertheless, since symmetric keys are shared, the sender must known the identity of the receiver, to authenticate before them.

For situations where the final destination of a message is known, HMAC can be used to prove the ownership of a key (e.g. Kerberos session key). However, since that is not always the case, there is a need for the HMAC architecture to be more flexible, by letting receivers request authenticated (with an HMAC) versions of the messages they receive. Nevertheless, this solution is not suitable for our target architecture, since there would be a significant increment in latency, not acceptable for the current requirements.

However, HMAC solutions could be considered as an extra security layer for situations where authentication is critical (i.e. commands with safety implica-

tions) or when the final destination is always known, but, as previously stated, they are not ideal for our target architecture.

Conversely, Kerberos delegation does not increment costs or management efforts and it does not affect latency. Moreover, it is very flexible, since the sender does not need to know the identity of the recipient and authentication is not message (end-to-end) but node-to-node based. The access servers are the nodes that decide where a message should be forwarded and authenticates the user against the destination or the next access server by impersonating them.

With Kerberos delegation, the trust lies on the access server(s), which can impersonate any recently connected users. As a result, authentication is only valid providing that access servers are not compromised. Therefore, even being the best solution in terms of flexibility and implementation effort, Kerberos delegation does not significant improve accountability, in comparison with the current architecture, where the user name is provided by the access server as part of the message.

In summary, digital signatures are the best solution in terms of provided security level, since they provide unequivocal authentication of the message source. However, PKI-based solutions would mean a high implementation and management cost.

Therefore, we consider digital signatures with Active Directory for credential management to be the most appropriate solution to our target architecture. Its main drawback is the latency when the operator logs in. If there is only one AD server it is not expected to be significant. However, when several AD servers are in place, synchronization would be needed, increasing the time window for the user's public key(s) to available to the receiving servers.

Nevertheless, as previously stated, digital signatures provide the greatest level of security and, if credential management is AD-based, they do not involve high implementation or maintenance costs, and they do not significant affect latency or CPU usage per message. We consider that these benefits motivate selecting this solution, even when the synchronization delay is considered.

7 Design

The solution selected based on the analysis (§6) relies on digital signatures for providing source authentication and employs the existing Active Directory server infrastructure for storing operators' public keys. This chapter will provide a detailed description of this solution.

The resulting architecture can be divided in two parts: in-band message signing and verification and out-of-band credential management. The former relies on digital signatures to provide integrity and source authentication, based on public keys. The latter allows operators to store their public keys, linked to their identities in the AD. Thus, it supports the authentication of operators by the receiving severs, based on the digital signature of their messages.

As with the current architecture, when an operator logs in to a workstation they are authenticated by the AD server using Kerberos. Once they are authenticated, operator applications generate a key pair and store their public key on an specific field in the AD server, whose writing permissions are limited to the owning user of the field. Thus, a public key in that field proves the authentication of the owner of the key by the AD server.

When a user sends a command to the access server, they sign an immutable part of the message with their private key, generated when logging in. When the destination receives the command request, forwarded by one or several access servers, they can authenticate the immutable part of the message by verifying its digital signature.

However, in order to associate a user with their signature, the server needs its public keys, available a the AD server. With those keys, the server can verify the digital signature of the operator. If the result is successful, the command can be executed and a log with the time, the command content and the identity of the user kept, providing accountability.

7.1 Message authentication and integrity

The in-band process of message signing and verification provides integrity and source authentication. When an operator sends a command, the message is signed using the operator's private key. The command request is then forwarded by one or several access servers.

In the current architecture, middle nodes do not necessarily preserve the message issued by the operator, but they typically generate new messages to satisfy user requests. However, in order to make authentication at the receiving end possible, the signed part of the message and its signature must be kept end-to-end. Otherwise, the receiving server would not be able to verify the digital signature issued by the operator.

Therefore, changes would be needed on the different server applications that perform changes on the commands messages. However, these applications differ depending on the particular command that is issued. Thus, although some of them may be tested during the implementation, the focus is put in the signature and verification process. Studying the upgrade of all of them falls out of the scope of this thesis.

Broadly, the upgrade process of middle applications would consist in having them include the signed part of the message and its signature in a field kept after subsequent modifications performed by middle nodes. According to ABB, this operation is possible in their current deployments.

Finally, when a SCADA server (or any other end server) receives a command request, it needs to read the signed part and verify that the signature was issued by the operator that allegedly requested the command. Should that be the case, the command would be executed and the identity of the user, together with the signed part of the command request, stored for accountability.

7.1.1 Signing

When a user issues a command request, they include a signed part that links the action they are requesting with their identity. That association is established by the use of a digital signature. As stated in §2.5.1, a digital signature proves that the issuer of a message is in possession of the private key linked to a particular public key. Thus, the presence of an electronic signature, provides source authentication for any receiving end that trusts the public key of the issuer.

The main challenge of applying digital signatures is deciding which is the payload to be signed. As previously stated, in the current architecture, messages are not necessarily preserved unchanged across the different nodes they traverse. On the contrary, access servers provide flexibility and scalability by interpreting commands and issuing new messages in order to execute de actions requested by the operators.

However, digital signatures are generated from a particular payload, which is the one that the issuer signs. Thus, if the payload is not preserved unchanged end-to-end, the verification of the signature would not be possible.

As a result, when operators issue a command, they need to append a signed text string that describes the command they are requesting. That data must be appended unchanged to any messages derived from that one, so when the end-servers receive the associated request(s) they can identify the original issuer and provide non-repudiation.

What is more, the alleged identity of the user and the public key used for signing must be also part of the message received by any end-server, so servers can obtain the public key associated with that identity that was used for signing and perform signature verification.

Moreover, the signed part of the message must not only include the description of the requested command, but additional metadata. Particularly, a time stamp and a nonce value must be present.

A time stamp allows any verifying node to discard delayed commands, that could lead to unexpected consequences. Whether they are caused by malicious agents (delay attacks) or by network errors, delayed commands may lead to unexpected results, especially in real-time environments as the ones SCADA servers commonly manage. Using a time stamp, receiving nodes can ignore commands issued after a threshold time.

In addition, a randomly created nonce (only-once) value provides receiving ends with a method for discarding repeated commands. Just as delayed messages, they can be caused by malicious parties (replay attack) or by the network. They may not be as hazardous as delayed commands but can also affect the functioning of the system and may be used for launching Denegation of Service (DoS) attacks.

7.1.2 Verification

The verification process is performed upon reception of a message by SCADA servers (or any other destination nodes). It provides source authentication and integrity protection of the signed part of the message.

When a message is received, the alleged user identity is obtained from the message and the public keys associated with that identity read from the system cache or from the credential management server. In any case, when using a public key for verification its creation and expiration date must be checked against the current time, to ensure the validity of the key.

If any of the cached public keys for the alleged sender matches the one on the message and it is within its validity period, that public key is selected. Should none of the cached keys matched the received one, the list of keys is updated from the AD.

Moreover, once the key contained in the message is confirmed to be associated with the alleged sender and within its validity period, it is used as the input for the signature verification process, together with the message and its signature.

If the verification process is not completed successfully with the updated list, or no key in the list matches the one in the message, that request can be logged and the security team alerted of a potential impersonation attempt.

On the contrary, if the digital signature matches the appropriate public key, the process continues. The time stamp is read and compared to the current time, in order to discard delayed messages. They may be logged but they cannot be considered as a an indicator of probable compromise, since they are more likely to be caused by network limitations.

Moreover, the command log must be queried for any messages that share time stamp and nonce with the received command request, in order to avoid executing repeated commands. As with delayed messages, they can be logged, but should not be consider as an indicator of likely compromise.

Once the previously mentioned steps are completed, the requested command can be executed and logged. Apart from a description of the actions performed by the SCADA server to fulfil the operator request and any extra information needed by the system administrators, the log entry must include the data included in the signed message: the time stamp, the nonce, and the description of the operator request.

On the one hand, the time stamp and the nonce are needed during the verification process for discarding repeated and delayed commands. Thus, the impact of network errors is mitigated and replay and delay attacks avoided.

On the other hand, the unchanged description of the command request, as issued by the operator, is needed for providing non-repudiation. Since authorization is managed by the access servers, if one of them is compromised the requests sent to the final servers may differ from the operator intention when initially issuing the command. However, even if the SCADA server runs that wrongfully modified command, the accountability logs would still store the original request issued by the operator, maintaining the non-repudiation property in the system.

7.2 Credential management

The previous section (§7.1) defines the process of in place for signing messages and verifying that signature. A digital signature provides authentication based on the ownership of a particular private key and its associated public key. However, the verifying node must be able to link that valid public key with an actual identity so it can provide operator authentication.

As stated before, a typical method for linking public keys with identities is the use of digital certificates. However, during the analysis chapter (§6) we decided to reject them for our design, due to their implementation and management cost. Consequently, we decided to rely on the existent Active Directory (AD) infrastructure for managing operators' public keys, linked to their identities on the AD server.

Asymmetric keys are designed to be in place for long periods of time (months or years). However, private keys should not be stored remotely and operators should not need to manage their key pairs. Thus, if an operator uses different workstations and keys are long term, one public key would need to be stored for each user and workstation. Moreover, in case of key compromise, revocation procedures would be needed for removing compromised keys.

Thus, in our solution key pairs are ephemeral, expiring after short periods of time (hours). Thus, since keys are short term, special revocation procedures are not needed, as keys they expire after hours, like Kerberos tickets.

However, operators may need to log in to various workstations at the same time (i.e. night shift staff). As a result, our design considers the possibility of simultaneously storing multiple keys for the same user. Keys would be removed by operator's applications when they log off.

Nevertheless, should sessions not be terminated graciously, old public keys may remain in the AD. If this situation occurs often, public key lists would become intractable, unnecessarily extending the verification process. Thus, a periodic purge of expired public keys in AD servers is recommended.

7.2.1 Credential management server and schema modifications

A trusted server is needed in order to store public keys linked to their owners identities. Since AD servers are already present at current deployments and they store user credentials, they were selected for managing public keys. Thus, our design is targeted to those servers. However, it would work with other

directory or database technologies (e.g. LDAP[7], SQL[8]), providing that they can authenticate operators.

If an AD server is used for storing user public keys, some changes on the Active Directory schema are needed. Particularly, the *User* class, whose objects represent a user of of the domain, needs to have an additional field. This field is a used to store operators' credentials.

Writing permissions of the public key field in the *User* object must be limited to the user that is represented by that object. Only by guaranteeing that just the user can write its own public key, the association between the public key and the user's identity can be trusted.

On the other hand, even if public keys were available to every domain user, security would be guaranteed, as public keys are design to be published. However, following the principle of least privilege, the reading access of the public keys should be limited to the owning user and to service accounts of the servers that verify digital signatures. Nevertheless, reading privileges may be extended if the proposed authentication scheme is used for purposes beyond the ones it is being designed for.

In addition, the creation date of each public key must also be recorded in the AD. An extra field could be created for this purpose, but to ensure the atomicity of writing operations, the same field is used in our design. The particular format for storing the key and the time stamp together will be discussed during the implementation phase.

The expiration date of the key could be used instead of the creation one. Nevertheless, the latter reduces the likeliness of implementation errors at the verifying nodes, since it enforces by design that the calculation of the valid time window is done by the verifying node.

Finally, it is common for organizations to have more than one AD server, establishing synchronization between them. This process can increment time needed for a public key to be available to verifying servers once a user publish it. During the implementation phase, our solution will be tested in different mock AD deployments and possible mitigations for this problem will be evaluated.

7.2.2 Signing node

In order to sign the messages they send, workstations need to have access to the operator's private keys. If the key pair is not cached in the operator workstation (i.e. the user has just logged in) or has expired, it needs to be generated.

During the implementation phase different asymmetric key algorithms will be tested. RSA is interesting due to its low verification time while ECDSA is fast for key generation and signing. However, regardless of the chosen algorithm, the overall functioning is the same.

Once the key is generated, the public key has to be uploaded to the appropriate field of the operator's *User* object. The creation date and time of the key must

[7]Lightweight Directory Access Protocol (LDAP)
[8]Structured Query Language (SQL)

be stored together with the key itself. As stated before, this time stamp provides a method for enforcing key expiration.

After the key pair has been generated and the public uploaded to the credential management server, messages can be signed with the private key. The key pair is removed after the expiration time is reached or when the operator logs off.

7.2.3 Verifying node

When a server (i.e. SCADA) receives a command request, the message signature must be validated and linked with the operator's identity. In order to perform this verification, the server must know the valid public keys of the command issuer.

Operators send their identities in the signed part of their messages. Thus, by reading them, verifying nodes can obtain the alleged identity of the command issuer. Messages also include the public key linked to the private key used for signing. This key must not be used for verification, but provides a fast method for identifying which public key from the operator key list is to be used for verification.

If no public keys are cached for that operator, they do not match the public key in the message or they have expired, the current list of public keys needs to be fetched. Once the list updated, any failure in the verification process must be considered final.

Regardless of the triggering event, the key fetching process is the same. Using the AD interface for accessing the AD data, the server requests the values of the public key list for the alleged user identity. Once downloaded, the list is parsed and each public key and its time stamp cached and used for the any pending verification operation.

Finally, once the list of public keys is fetched, each key can be used unlimitedly during its validity period. Updating the list is only necessary when verification fails or if all the keys expire.

7.3 Requirement fulfilment

7.3.1 Architecture constraints

Our solution can be implemented in the current architecture with no significant modifications. Commands are still requested using Remote Procedure Calls (RPCs) and node-to-node connections protected by Kerberos tunnels.

The operator account is only used for connecting their workstation with the first access server. However, as with the existent architecture, when access servers forward command request they use service accounts for authentication. On top of that existing architecture, our design provides end-to-end operator authentication and non-repudiation.

Firstly, the Active Directory schema needs to be modified to be able to store operators' public keys, guaranteeing that the link between operator identities

and their public keys is trustworthy.

Moreover, verifying and signing nodes also require modifications. The former need to be able to read public keys from the AD server and to use them to validate digital signatures. They also have to store into a log the issuer identity and the request description for each command to provide non-repudiation. Similarly, signing nodes must create key pairs, store public keys in the appropriate field at the AD server and include a signed part in command requests.

Finally, in the current architecture, command requests are likely to be modified in transit by access servers. This behaviour is still present, but the signature and its payload must be maintained unchanged end-to-end. ABB current protocols provide methods for introducing additional data (i.e. text strings) in RPCs. Using them, it is possible to provide the signature information to the verifying nodes with no important architectural changes.

7.3.2 Security requirements

The existent architecture ensures authentication, confidentiality and integrity in node-to-node connections. As a result, when command requests are forwarded, the trust lies in the forwarding nodes (access servers), that inform about the requesting operator identity.

The main purpose of our design is to provide authentication end-to-end, independent of the middle servers. Thus, even if an access server is compromised, SCADA servers (or any other receiving nodes) can authenticate the command issuer and provide non-repudiation.

However, in current deployments, command authorization is not managed by the SCADA server itself, but by access servers. They authenticate and authorize commands, by deciding if they are forwarded or discarded, depending on the requested action and on the permissions of the requesting operator. SCADA servers only authenticate incoming commands, but they do not provide authorization.

Moving command authorization from access servers to SCADA servers may be possible. Nevertheless, it falls out of the scope of this thesis. It would also reduce the capabilities of access servers and, consequently, the flexibility of the whole architecture.

Thus, our design does not ensure proper command authorization in case of access server compromise. As with the current architecture, access servers authenticate and authorize commands, while the SCADA server only guarantees that command issuers are authenticated.

Therefore, should an access server be compromised, command-request authentication would still be valid, but privilege escalation attacks would be possible. In other words, if a low-privilege operator manages to compromise an access server, they would be able run commands without being authorized. However, even if that were the case, their actions would be logged, linked to their authenticated identities.

Moreover, confidentiality is guaranteed in node-to-node connections by the Kerberos tunnels already used by the current architecture. In addition, our design

provides end-to-end integrity protection for the signed parts of the command requests.

In addition, our solution also provides additional protection against replay or delay attacks. By including a time stamp and a nonce in the signed part of the command requests, delayed or replayed messages can be detected, logged and discarded.

Finally, with this design, trust lies in the AD server. Thus, domain administrators can impersonate any user by adding their own public key to the AD. This risk can be mitigated by logging administrator actions, but not eliminated. As discussed during the analysis, all solutions trust a credential management server to some extent, and are vulnerable to its compromise.

7.3.3 Performance requirements

The established performance requirements state that the CPU usage and latency impact of our solution must be as low as possible.

In terms of CPU usage, the effect of deploying our design is not expected to be significant. Signing and verifying nodes would be the most affected ones, due to the computational cost of asymmetric-key cryptography operations. However, as denoted during the analysis (§6.4), modern computers can manage asymmetric-key cryptography with no significant impact in computational resource utilization.

Moreover, standard key length requirements for public-key cryptography are established for long term use of the keys. Thus, key length recommendations, issued by advising bodies, can be relaxed in our design. A high level of security would still be provided, due to the ephemeral nature of the keys in our solution.

Similarly, latency impact per message would be affected by signing and verification operations. Nevertheless, modern computers can perform asymmetric-key operations, not only with low CPU usage impact, but also with insignificant effects in latency.

However, our design may affect initial latency after an operator logs in. In infrastructures with multiple AD servers, synchronization operations must be performed, in order for directory data to be consistent across all AD servers. These operations can take several minutes. Thus, since the operator's public key stored in the AD directory must be available for the verifying nodes, the time window since an operator logs in and until they are able to send commands can be severely affected.

During the implementation stage, the effect of this initial latency will be evaluated. Moreover, mitigations for these potential delays will be explored.

8 Implementation

In order to prove the validity of design, a prototype implementation has been produced. This chapter discusses how it was created, covering the installation scripts for the AD Server and the applications for the verifying and signing nodes.

8.1 AD Server

8.1.1 Adding a field for storing operator public keys

By default, AD controllers do not allow the modification of the schema. In order to be able to do so, a key must be added to the Windows registry on the AD domain controller.

Afterwards, the attribute for storing the public keys must be added. We decided that the attribute must be multi-valued, so it can store several public keys. Moreover, keys can be stored as text strings (ASCII, Unicode). However, binary formats require less space and remove the need for conversion. Thus, since users do not need to read the keys themselves, they are directly stored in binary format, using the AD type *octet string*.

Once this attribute is created, it needs to be part of the *user* class, which represents a user of the domain. Nevertheless, instead of adding it directly to the class, an auxiliary class is created, and the attribute added to it. Then, the new class is set as auxiliary for the *user* class, that inherits the attribute.

Having the auxiliary class as owner of the attribute provides more flexibility if further modifications are required. Moreover, the class is an indicator of the changes on the schema, helping in potential subsequent modifications or audits.

In addition, Active Directory elements (attributes and classes) are identified by and Object IDentifier (OID). OIDs can be divided in two parts: a base OID that identifies the owning organization; and the element OID, linked to a particular element of that organization. Base OIDs can be requested from Microsoft for testing purposes; but, for production environments, they should be issued by a ISO-member organization [59].

Furthermore, all custom fields in the AD schema must have a prefix that identifies the owning organization. It provides a method for easily identifying the owner of the custom fields, without needing to verify the base OID registration.

A PowerShell script is provided for performing the previously explained operations. It enables schema modifications, creates the attribute and the auxiliary class in the AD, and sets the new class to be auxiliary of the system-default *user* class.

The script requires a prefix and a base OID as parameters, in order to apply them to the class and attribute to be created. Moreover, it also accepts the element OID for the new class and attribute as parameters. If they are not passed, 1.1 is used for identifying the new class and 2.1 for the attribute.

8.1.2 Writing permissions of the public key field

By default, only domain administrators can modify the new attribute of the *user* class. In order for our solution to work, each user must be able to write its own public key on the public key field of its user object.

Domain administrators must delegate the control over the new attribute to the users. The *SELF* user represents the user linked to a particular *user* object. By giving writing permissions to *SELF*, each user is be able to write or delete its public keys on the AD, but not the ones of the rest of users.

However, domain administrators are still able to write in the public key field of every user. By design, AD administrators have full privileges over their domain. Thus, the best way to mitigate the risks involved with this approach is to limit the number of domain administrators (least privilege policies), protect their credentials, and log all their actions in the system.

8.1.3 Reading permissions of the public key field

By default, reading permissions are granted to all authenticated users. Even though this is enough for guaranteeing the validity of digital signatures, it should be extended as defined in the design chapter. Particularly, reading permissions for the public key should be limited to the owning user and service accounts.

For implementing this policy, a general rule that blocks reading attempts to every user must be set, together with a more specific rule that allows access to particular users. The main limitation of the AD for this approach is that *deny* rules take precedence over *allow* ones.

However, rules set higher in the hierarchy are superseded by those defined in lower levels. Thus, a workaround for this constraint lies in setting the *deny* rule on a AD element higher in the hierarchy than the one where the *allow* rule is set. For instance, if the *allow* rule is set on a User container, the *deny* rule can be established on the Organizational Unit (OU) or on the domain.

Following these guidelines, a *deny* rule must be set for the group *Everyone*, that represents all the domain users (including those who are not authenticated). Then, *allow* rules must be defined for the user that owns the public key field and for the service accounts. The former can be established while setting writing permissions, while the latter should be applied to a group of users that contains all the service accounts.

8.1.4 Purging obsolete keys

Even though the signing application removes keys from the AD when the user logs out, sessions may not always be closed graciously. Should that be the case, expired keys would remain in the AD. Their presence would not have any security implications, since they would be discarded by the verifying node when checking its time validity. However, a great number of keys per user would lead to an increment in the time needed to find the correct public key.

In order to relieve signing nodes from checking the state of the key list for their

users, the server can periodically remove expired keys. The frequency of the key purging operation may vary depending on the particular characteristics of each deployment, being a weekly run appropriate in most cases: It is frequent enough to avoid long key lists, while it can be run on server valley times for avoiding unwanted effects in other servers.

A Powershell script *CleanExpiredPublicKeys.ps1* is provided. When executed, it verifies all public keys for all domain users and removes those which are not within their validity period. This script can be set as a task in the domain controller to be run periodically by the system.

8.1.5 Domain controller replication

In order to provide redundancy or for sharing the same AD infrastructure among different sites, AD replication can be established. Depending on the the location of the controllers and on the configuration, synchronization times may vary from seconds to hours.

Should operators connect to a different controller than receiving nodes (i.e. SCADA) server, replication delays may have a significant effect in the latency of our solution. However, effects differ depending on the configuration of the controllers. For instance, if they are logically located in the same site, replication of changes has a latency of approximately 15 seconds.

On the other hand, if controllers are configured in different logical sites, synchronization is done every 3 hours by default. This frequency can be modified, but the minimum value is 15 minutes. Obviously, a 15 minute delay since a user logs until the receiving end is able to verify their messages is not acceptable.

One alternative to address this limitation relies on modifying the configuration of the link between sites. The *USE_NOTIFY* option can be switched on. If enabled, domain controllers issue notifications when changes are performed on the AD, triggering a synchronization with the rest of the controllers.

Thus, should the *USE_NOTIFY* option be enabled, changes would be propagated as if nodes were in the same logical site, having a latency after login of a approximately 15 seconds (without considering link delays). However, since this option is not targeted to a particular AD class or attribute, replication would be triggered with every AD modification. Therefore, in cases where the link between sites has bandwidth limitations, it could lead to network congestion and create inconsistencies between controllers.

As a result, another option has been explored. It relies on triggering AD replication with log events. By enabling auditing on the public-keys field of the user objects, an event is generated every time a key is written for a user.

Windows events can be used, not only for auditing purposes, but also for triggering tasks. Particularly, a task can be triggered by write-access events on the active directory. These events include a field called *Properties* that has three Globally Unique IDentifiers (GUIDs). Two of them are default identifiers, that do not differ depending on the object or attribute modified. However, the other one identifies the attribute that has been modified.

As a result, selecting only those events that have the appropriate GUID is

crucial for running the replication process only when needed. Nevertheless, Windows event filtering is based on a limited version of XML Path Language (XPath), that only filters strings with exact matches. Moreover, the properties field includes hidden characters (e.g. line breaks) that are difficult to match, and whose syntax may vary depending on the system.

Therefore, a task is triggered for every write operation on the AD, regardless of which attribute was written. It executes a PowerShell script that receives the event information and decides whether the event represents the writing of a public key on a user object. If that is the case, a replication operation will be executed for that user object.

This solution is more complex in terms of implementation than the *USE_NOTIFY* switch. Furthermore, it could affect log size if many write operations are performed on the AD. However, it provides a finer-grained triggering for synchronization, avoiding the link congestion that too many replication operations may cause.

8.2 Signing node

Our implementation for the signing node provides signing of commands and manages the associated public key in the Active Directory. Moreover, a simple socket client was implemented for simulating the communication with the verification node.

Furthermore, at ABB, most operator workstations run Windows. Thus, our implementation for these nodes is targeted for that operating system. Particularly, it relies on the .NET Framework and the C# language.

The .NET Framework is installed by default in the latest Windows version, and available for installation in older versions of the operating system. It provides Graphical User Interface (GUI) capabilities by the use of forms (Windows Forms) and includes implementations of cryptography functions.

Two versions of the client application have been implemented, one that uses ECDSA and another that relies in RSA. Having different applications for each signing algorithm facilitates the enforcement of one of them, by distributing the appropriate application to clients.

Moreover, the support of ECDSA in older versions of the .NET Framework is limited. Thus the ECDSA variant of our application needs at least version 4.7 of the .NET Framework. On the other hand, the RSA variant is compatible with version 4.0 and higher. Therefore, should workstations have obsolete versions of the .NET Framework, the RSA variant of the client application can be used.

As stated before, the client application relies in Windows Forms to provide a graphical interface. Particularly, the application GUI is divided into three forms:

- Login. Key pair generation and uploading of the public key to the Active Directory
- Connection. Binding to the server running at the verifying node

- Messaging. Sending of signed messages to the server

8.2.1 Active Directory connection (Login)

The *Login* window has only one button for starting the login operation. As Figure 8 shows, this form also states the identity under which the user will be logged in. When the user logs in, their public key is stored in the AD, linked to their identity. Conversely, when a user logs out, the public key for that session is removed from the AD and the local object that contains discarded.

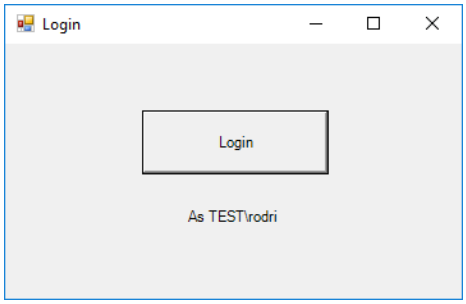

Figure 8: Login form of the signing application

Internally, the application uses an interface that exposes the methods use by

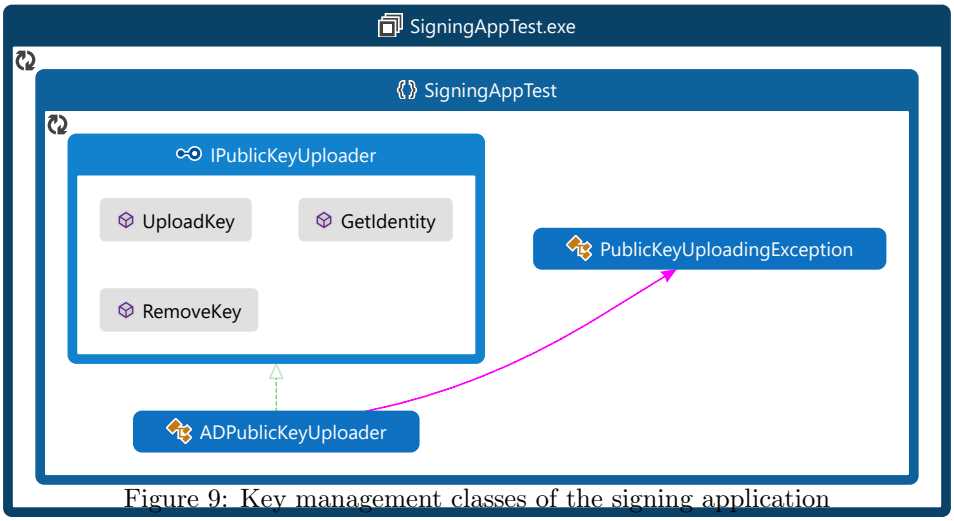

Figure 9: Key management classes of the signing application

The use of an interface provides an easy method for replacing the key management method, since the login and signing process would work with any valid implementation of the interface. Our particular implementation relies on the active directory for storing the public keys, as discussed in the design chapter (§7).

The identity of the user is not its user name, but its Security IDentifier (SID). Using an unique identifier guarantees that the the public key is stored for the

Time stamp (8 bytes)	Key data (Microsoft BLOB) (n bytes)

Figure 10: Common public key format

Time stamp (8 bytes)	Key type (4 bytes)	Key size (4 bytes)	X (32 bytes)	Y (32 bytes)

Figure 11: Public key format (ECDSA with curve NIST P-256)

correct user. It also forces the verifying node to obtain the human-readable user identity from the AD for logging purposes, ensuring its correctness.

For adding or removing a key to a user, the related object must be retrieved from the AD. The .NET library includes tools for performing such operation (*System.DirectoryServices*). By providing the user SID, the related directory entry can be obtained, and keys added or removed.

Due to the usage of the .NET library for signing messages, the public key can only be exported in Microsoft's proprietary format. Additionally, when the *SignatureManager* exports a key, it includes its creation time stamp. It is stored in milliseconds since the (Unix) epoch in a 64-bit integer, stored in big-endian format (the most significant bit is stored first).

As discussed during the design chapter (§7), having a time stamp provides a method for discarding old keys. The general structure of exported public keys, common for RSA and ECDSA, is shown in Figure 10.

ECDSA key format When using ECDSA, the key is expressed as the coordinates of a point in the elliptic curve. It is exported together with an identifier of the key type, and a value that represents the length of each coordinate of the key. Figure 11 shows how the data is formatted when ECDSA keys are stored in the AD.

Our implementation relies on the P-256 elliptic curve, defined by the NIST [60]. As discussed in §2.4.3, it provides a high level of security, even for long-term keys. Since in our design keys are expected to last for short periods of time, the architecture would still be secure with shorter ECDSA keys.

However, the selected curve was chosen because it is compatible both with the .NET Framework and with the Python ECDSA library of the verifying application. Moreover, according to the ECRYPT II benchmarks [7], it is the fastest ECDSA curve in terms of key generation, signing and verification; even when compared to shorter ones.

RSA key format In order to be compatible with older .NET Framework versions, and to compare the performance of RSA and ECDSA, we also provide an RSA based implementation of the signing application. Key generation and message signing is expected to be slower than with ECDSA, but verification faster, reducing the load of the verifying node.

Time stamp	PUBLICKEYSTRUCT	RSAPUBKEY	modulus
(8 bytes)	(8 bytes)	(12 bytes)	(1024 bits / 128 bytes)

Figure 12: Public key format (RSA 1024 bit)

BLOB type	BLOB version	reserved	Algortihm ID	Padding
0x06	0x02	0x00	0x0000a400	0x00
(1 byte)	(1 byte)	(1 byte)	(4 byte)	(1 byte)

Figure 13: Public key format (RSA 1024 bit): PUBLICKEYSTRUCT

The short term nature of our keys reduces the time window for brute-force attacks. Thus, even though RSA-3072 could be considered an equivalent of ECDSA P-256, we have selected RSA-1024. Using shorter keys, reduces the computational cost of cryptographic operations.

Moreover, as stated in §2.4.4, the longer RSA key to be broken was a 786 bit one, and it took several years. Thus, we consider that a RSA 1024 bit key pair is more than enough for our architecture, since keys will expire in days or hours.

RSA keys are also stored in the AD with their time stamp, according to the format defined in Figure 10. However, the .NET RSA library exports keys with more metadata than its ECDSA counterpart. It includes a *PUBLICKEYSTRUCT* structure, that identifies the blob as an RSA public key blob; a *RSAPUBKEY* structure, that defines the public key format; and the modulus. Figures 12,13 and 14 cover the format of RSA public keys stored in the AD and sent over the network.

8.2.2 Socket client (Connect)

Our implementation is targeted for ABB proprietary communication protocols. However, since our focus is on message signing and verification, setting up our implementation for all ABB communication protocols falls out of the scope of this thesis.

Thus, a simple client-server architecture has been created for modelling message passing between nodes. Particularly, the signing node implements a socket-based TCP client, that connects to the receiving end for exchanging messages.

Connections are started by the user from the *Connect* form (Figure 15). The user needs to write the server address and port to start a TCP connection to it.

Key type (Magic ID)	Bit count	Public exponent
0x52534131	1024 (0x00040000)	[Dynamic value]
(4 byte)	(4 byte)	(4 byte)

Figure 14: Public key format (RSA 1024 bit): RSAPUBKEY

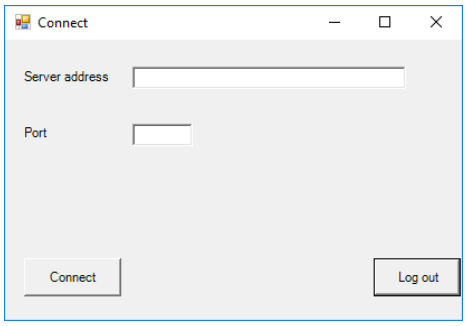

Figure 15: Connect form of the signing application

Internally, the socket server works asynchronously, guaranteeing that the interface stays responsive, even when the application is working on the background. When a connection is established, the GUI switches to the *Messaging* form, that provides message sending capabilities.

8.2.3 Signature (Messaging)

Once the user's public key is stored in the AD controller and a connection with the receiving server has been established, signed messages can be sent to the server from the *Messaging* form (Figure 16). It provides a text box for sending messages and shows information about what is sent to the server and its response.

Internally, the *SignatureManager* class provides the means to sign messages, relying on the .NET cryptographic libraries. It also stores the key pair used for issuing signed messages and provides access to the public key.

When the user wants to send a message, a the signature payload is created. It includes the message (the command description in the target architecture), a time stamp, a nonce and the identity of the sending user. For compatibility reasons, the time stamp is stored in milliseconds since the (Unix) epoch. Moreover, the identity is the user Security IDentifier (SID) of the sender.

Afterwards, the signature payload is converted to a JavaScript Object Notation (JSON) and signed with the private key. The JSON format was chosen for being compatible with most programming languages. Moreover, being human readable aids in understanding the functioning of the application when sending and receiving messages. Nevertheless, in order to increase performance, a byte based format may be created, reducing the size of the data sent over the network and removing the need of performing JSON parsing operations.

Subsequently, a signed message is sent to the receiving end. It is a JSON object that includes the signature payload, the signature, and the associated public key. The public key is sent without its time stamp, since it is only needed for selecting the appropriate key from the AD, where keys have the time stamp appended.

Finally, the application waits for the response from the receiving node. Once received, it is shown in the GUI to the user (Figure 16).

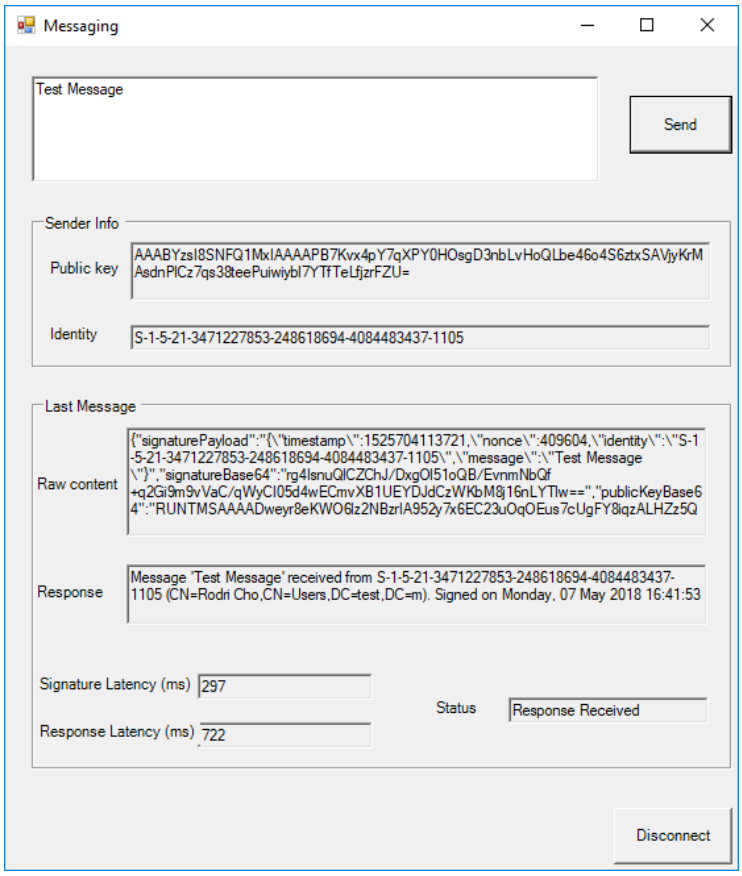

Figure 16: Messaging form of the signing application

8.3 Verifying node

The verification application is a socket server that accepts connections from clients. Once client is connected to the server, their messages messages are parsed. Particularly, the sender alleged identity and public key are read and compared against the keys for that user in the Active Directory. Once the valid public key is retrieved, it is used to verify the signature payload. Finally a response is sent to the client, indicating if the message was validated.

Python was selected for implementing this application, due to its simplicity and compatibility with most Linux systems. Moreover, there are Python libraries for connecting to the AD (LDAP protocol) and implementations of cryptographic functions (RSA and ECDSA). Although Python 3.6 version was selected initially, due to compatibility limitations in ABB target systems, Python 2.7 is the final choice. Furthermore, since ECDSA and RSA libraries were not installed in ABB test system, we include them with our implementation.

In addition, even though we consider Python a good choice for demonstrating our design, real implementations could benefit from using more efficient programming languages, such as C/C++. They offer better performance, since

they provide a finer-grained computing resource management and are compiled before running.

8.3.1 Key retrieval from the AD

The *key_provider.py* module exposes the *PublicKeyManager* class, for providing access to the AD public keys. Particularly, it stores a list of keys for each user in a dictionary and updates it when needing by connecting to the AD.

The *get_validated_public_key* method, receives the alleged sender's identity and public key of a message and tries to find it on the cache. If the key is not found the cache entry for that identity is updated, and the process repeated again. If no valid key is found or the key is not within its validity period, an exception is thrown. Otherwise, the valid key is returned, with the time stamp data removed.

The connection to the Active Directory is managed by the *ADPublicKeyProvider* class, that binds to the selected domain controller when instantiated (when the program starts running). The credentials for the bind operation are requested to the user running the application, while the rest of parameters needed (LDAP address of the domain controller, domain name and the name of the attribute that stores the list of public keys) are read from a JSON configuration file.

When a list of keys is requested to an *ADPublicKeyProvider* object, the associated user element is retrieved from the AD, using its Security IDentifier (SID) as a filter. From that object, the list of keys for the user is return, together with the human-readable name of the user. Even though the SID is better for performing searches due to its uniqueness, we consider that the name of the user is more interesting for logging purposes, since it provides a faster identification of the sender when reading the logs.

8.3.2 Key parsing

Regardless of the cryptographic algorithm (RSA or ECDSA), the Python libraries our implementation relies on do not support the Microsoft BLOB format used by the signing applications. Thus, a conversion must be made before running any cryptographic operations.

Firstly, algorithm needs to be identified. Currently, our server only supports RSA and ECDSA P-256 . Once a key is validated, its headers are checked for deciding which algorithm is being used.

ECDSA keys are recognized by their key type field, a magic ID define by Microsoft. Particularly, P-256 keys have the 0x31534345 value that represents the *BCRYPT_ECDH_PUBLIC_P256_MAGIC* type and is stored in little-endian format (bytes in reverse order).

Once an ECDSA P-256 key is recognised, it is expected to have a key size value of 32 (32 bytes = 256 bit). Then, the coordinates of the key (a point in the curve) are read and used for generating a verifying-key object.

Conversely, RSA keys are identified by their *PUBLICKEYSTRUCT* structure (Figure 13) and the key type in the *RSAPUBKEY* structure (Figure 14).

Verifying-key objects for RSA are created by reading their bit count and public exponent. The modulus length is calculated depending on the value on the bit count field. Thus, even though it has not been tested, it should work with any key lengths compatible with the Python RSA library in place.

Finally, regardless of the algorithm, verifying-key objects (from the *CustomECDSAVerifyingKey* or *CustomRSAVerifyingKey* classes) expose a *verify* method that receives a message and a signature and raises an exception if verification fails.

8.3.3 Validation of signed messages

When a message is received from the server, it is passed to the *verifyMessage* method of *VerifyingManager* object, created when the application is started.

The JSON data that contains the signed message information is parsed, obtaining the signature payload, the signature, and the alleged public key. The payload has to be parsed too for retrieving the alleged user identity.

Afterwards, the key is validated, by calling the *get_validated_public_key* method of the *PublicKeyManager*. If the key provided by the user is validated (in terms of validity time and presence in the AD), it is used for creating a verifying-key object, as described in the previous section (§8.3.2).

Finally, using that object, the signature payload is verified. A message is sent back to the client, stating whether the verification process was successful. On the contrary, should errors occur during this process, the response would include information about the particular error cause.

9 Performance evaluation

Apart from proving the viability of our design, the implementation also provides a means for producing a limited performance evaluation. Particularly, we measured the time needed by ABB testing systems for performing cryptographic operations. Moreover, the latency for message passing and for synchronization of the list of public keys between Active Directory controllers has also been studied.

The test environment consists in one Windows client, one Linux server (i.e. the SCADA server) and two AD domain controllers. Table 7 shows the specifications of those systems.

All of them run in virtual machines managed by VMware's ESXi, a type-1 hypervisor[9]. Type-1 hypervisors (also called native or bare-metal), are those that run directly on hardware, controlling it and having more flexibility when managing guest systems.

9.1 Cryptographic operations

Even though modern processors have reduced the time needed for performing cryptographic operations, their impact must be studied; not only for assessing the potential deployment of this solution on ABB systems, but also for comparing the performance of RSA and ECDSA. Measurements were obtained in terms of time needed for performing cryptographic operations in ABB test environment.

9.1.1 Key generation (Signing node)

When a user logs in, the application generates a key pair and uploads the public key to the AD. The time required for performing key generation, together with AD replication times in some cases, is important for predicting login times.

Table 8 presents the mean, median and standard deviation for the time measurements retrieved after 10000 key generation operations. Values where measured using .NET *Stopwatch* function.

[9]**hypervisor**: Virtual machine monitor. Creates and runs virtual machines.

Table 7: Testing system technical specifications

	Signing node	Domain controller	Verifying node
Architecture	64 bit	64 bit	64 bit
Processor	2 x 2.60Hz	2 x 2.60Hz	2 x 2.60Hz
RAM	4 GB	2 GB	8 GB
Operating system	Windows 10 (1607)	Windows Server 2012 R2	Red Hat Enterprise Linux Server 7.3 (Maipo)

Table 8: Key generation time in milliseconds (10000-element sample)

	Mean	Median	Standard deviation
ECDSA P-256	0.2145	0.2159	0.0616
RSA 1024 bit	11.013	9.4483	6.5899

Table 9: Message hashing and signing time in milliseconds for 100-byte messages (10000-element sample)

	Mean	Median	Standard deviation
ECDSA P-256	0.1199	0.1097	0.0265
RSA 1024 bit	0.2834	0.2718	0.0044

9.1.2 Signature generation (Signing node)

Every message that a user sends to the verifying node must be signed. Thus, the time needed for performing signing operations must be measured, since it affects the latency of every command request issued by the operator. These measurements include not only the signing of the message hash, but also the generation of that hash digest. Separate measurements are not available, since libraries perform both operations within the same function call. Table 9 presents the mean, median and standard deviation for the time measurements retrieved after 1000 key generation operations.

9.1.3 Signature verification (Verifying node)

Every message received at the verifying node must be verified, to decide whether it is valid. Just as signature generation, verification operations affect the latency of every sent message. Moreover, Python libraries also perform hashing of the message contents and signature verification within the same function. Thus, values presented in Table 10 measure both operations as if they were one. Times were measured using the *time* function of the Python *time* module, that provides accurate system clock readings in Linux systems.

9.2 Message passing

Latency in key upload and retrieval depends mainly on the network that connects the nodes and on protocols used by ABB to forward command request and responses. Thus, message passing latency is greatly affected by factors that are external to the application and can vary from one particular target architecture to another.

Table 10: Message hashing and verification time in milliseconds for 100-byte messages (1000-element sample)

	Mean	Median	Standard deviation
ECDSA P-256	89.462	84.824	14.195
RSA 1024 bit	0.1354	0.1221	0.0383

Table 11: Active Directory public key propagation times

AD environment			Propagation/Replication time
Controller	Site	Configuration	
Same	-	Default	0s
Different	Same	Default	15s
	Different	*USE_NOTIFY*	15s
	Different	Default	180 min. (minimum 15 min.)
	Any	Event-triggered replication	4 ± 1 s (6 ± 1 s after boot)

Moreover in our prototype message passing is done using a socket client-server architecture, different from ABB RPC protocols. Therefore, the latency of the communication in the test architecture is not applicable to the production environment.

Nevertheless, we can measure the overhead that our application produces when sending signed messages. It provides any potential user of our solution a method for predicting latency in their own architecture. Results showed that the overhead is of 522 ± 2 bytes per message when using RSA-1024 and 334 ± 2 when using ECDSA P-256.

9.3 Key retrieval delay

As with message passing, measurements of key retrieval and storing latency would not provide significant data, as they depend on the network and on the Active Directory communication protocols.

Nevertheless, when the AD domain controller is not shared by the signing and verifying nodes, the delay since a user logs in (uploads it public key) until their key is available for verifying nodes does not depend only on network delays.

On the contrary, they are mainly dependant on the Active Directory replication times. These times have not been measured exactly, since they are affected by the inherent unpredictability of network environments and are therefore hard to measure confidently. However, default values were obtained from bibliography and those that were not available were measured producing approximate values.

When domain controllers are logically located on the same site, their default synchronization time is of 15 seconds [61]. On the other hand, when controllers are configured in different sites, their replication times are longer: by default 180 minutes, being possible to reduce it to no less than 15 minutes.

In §8.1.5 we proposed two methods for reducing this delay, namely setting the *USE_NOTIFY* option or using the event log to trigger targeted replication. The former needs 15 seconds for replication [61], as measurements confirmed. Conversely, event-triggered replication was faster than the *USE_NOTIFY* option, making the key available in 4 ± 1 seconds, except the first time after system boot, when the delay was of 6 ± 1 seconds.

10 Discussion

The design and consequent implementation presented in this thesis provide a solution that fulfils the requirements established in §5. This chapter explains how these requirements are meet in terms of compatibility with the target architecture, security and performance.

10.1 Compatibility with the target architecture

Our design is compatible with the target architecture with minimal deployment costs. Moreover, our implementation has not been tested on real production environments. However, ABB RPC protocols provide methods for sending text content attached to command requests. Thus, since communication between signing and verifying node is message based, our implementation can be applied to the target architecture by using the text field in ABB protocols as a means of communication between nodes.

Moreover, changes would be needed on the applications at the workstations and at the receiving nodes (e.g. SCADA servers), since they need to be able to sign and verify messages respectively. Furthermore, Active Directory schema must be modified in order to provide identity management to our solution.

Our implementation has proven the viability of applying our design in terms of signing and verifying messages and using the AD for identity management. Moreover, there is practically no extra maintenance cost, as key purging operations can be done automatically.

Furthermore, there is no need for extra credential management, since identities are managed by the Active Directory as they were before and the short term nature of our key pairs removes the need for credential revocation in case of key compromise (although it could be done if needed).

Finally our implementation does not incur in extra costs, as it does not need extra hardware and it is based on software that is already present on the infrastructure or that is freely available.

10.2 Security

Our solution relies on digital signatures for providing message source authentication and non-repudiation, together with Active Directory for linking identities and public keys. Moreover, time stamps and nonces are used for avoiding replay and delay attacks.

10.2.1 Digital signatures

As stated in §2.5.1, digital signatures are the most common cryptographic solution for guaranteeing authentication and non-repudiation. The level of security they provide is mainly dependant on the public key algorithm used for generating and verifying them. In our implementation, we selected ECDSA P-256 and RSA 1024 bit. The former is recommended by the ECRYPT group as appro-

priate for long term usage, and thus it exceeds the security requirements of our solution (keys are short term).

Conversely, both the NIST and the ECRYPT group state that RSA keys must be 3072 bit long for long term applications. However, since our keys are to be valid only for hours or days, requirements can be relaxed, as the time window for brute force attacks is reduced in comparison with long term usage. Particularly, we selected a 1024 bit length because, while it provides better performance, the longest RSA key to be broken was 768 bit long (and researchers needed several years to break it).

In any case, this particular choices of key lengths for RSA and ECDSA may become obsolete over time, as computing evolves. Thus, they will need to be updated accordingly to maintain the security of the solution. Particularly, it is important to keep track of the advances in quantum computing, as these technology could be able to broke both algorithms in the future. Should that be the case, alternative algorithms could replace RSA or ECDSA, as discussed in §2.4.4.

10.2.2 Active Directory for identity management

In order to reduce implementation and maintenance costs, the Active Directory is used for storing user public keys. This directory is already present in the target architecture and manages user identities. Thus, as proved during the implementation, it can be used for storing the public keys needed by our solution with minor modifications on its schema.

The use of the AD for storing public keys has already been explored by system administrators. Particularly, several online resources reported having used it for storing SSH public keys [62][63][64] or PGP public keys [65].

However, relying on the Active Directory for credential management of digital signatures has a downside: It becomes a Single Point Of Failure (SPOF) in terms of attacks. If any of the controllers is compromised, an attacker would be able to write its own public keys and use the credentials of any user on the domain. Thus, protecting Active Directory domain controllers must be a priority for maintaining the security of the architecture.

Moreover, domain administrators have virtually no limitations to write in the AD. Thus, they can write their own public keys on any user, consequently being able to impersonate them. Therefore, their accounts must be specially protected. In addition, logging of writing events on the AD can be enabled, not only for protection against hacked accounts, but also for detecting malicious actions by rogue employees.

10.2.3 Peculiarities of the target system

Communications are already protected by Kerberos tunnels in the target architecture, guaranteeing node-to-node confidentiality, authentication and integrity protection. On top of that architecture, our solution provides end-to-end authentication and integrity for the signed messages.

The authentication guaranteed by digital signatures, together with the identity management provided by the Active Directory, ensures non-repudiation of command requests, since they can be linked to a particular user. Since authentication is provided end-to-end, this association between user and command request is to be trusted even in the event of access server compromise.

However, as stated in §7.3.2, our solution does not provide authorization of commands. The evaluation of the privileges when a user runs a command is performed by the access servers, which forward or discard a command depending on the permissions of the user.

Thus, as with the current architecture, command authorization in our solution depends on the access servers.They authenticate and authorize commands, while the SCADA server only guarantees that command issuers are authenticated.

Therefore, should a low-privilege operator manage to compromise an access server, they would be able run commands without being authorized. Nevertheless, even if that were the case, their actions would be logged, linked to their authenticated identities.

Moving command authorization from access servers to SCADA servers would be a solution for this limitation. Nevertheless, it falls out of the scope of this thesis. It would also reduce the capabilities of access servers and, consequently, the flexibility of the whole architecture.

10.2.4 Additional security considerations

In §5, we established that the protection against replay and delay attacks was a requirement for our design. In order to fulfil that requisite, our signed messages include a time stamp and nonce. The former provides the creation time of the message to the receiver, enabling the rejection of delayed messages and consequently protecting against delay attacks

On the other hand, the nonce (a random value to be used only once), ensures that every message sent by the client is different to the rest. Therefore it allows the receiver to identify repeated messages by comparing them with the logged ones.

An attacker may try to avoid this security measure by sending a command to a SCADA server different from the one that received the message. The potential impact of this attack depends on the particular distribution of functions per SCADA server. In any case, attacks would be limited to a particular time window (i.e. until command requests expire) and would need internal access to the system, as communications are encrypted node-to-node. Moreover, centralised log parsing could help to identify message reception patterns potentially related with replay attacks.

10.3 Performance

The performance evaluation (§9) compared the efficiency of the tested cryptographic algorithms (RSA and ECDSA) and the overhead per message of the implementation. Based on that, this section discusses about which algorithm

would be more appropriate depending on the particular characteristics of the target architecture. Moreover, different options for domain controller synchronization of public keys are commented, relying on the performance evaluation results.

Nevertheless, in order to fully evaluate the performance impact of our design in the target architecture, more parameters should be measured, such as the computational capabilities of the nodes or the properties of ABB RPC protocols. Since these values vary depending on the particular architecture, measuring the falls out of the scope of this thesis and our performance evaluation is limited. However, it provides illustrative results that can be used for assessing the suitability of our solution for a particular target architecture, aiding in the decision making process.

10.3.1 Cryptographic algorithms

It is important to consider that our comparison is not fair, since the the ECDSA curve that we are using provides a higher level of security than the RSA key length set for the evaluation operations. However, our purpose is not to evaluate these algorithms, but the performance of our application when using them. That is why we are using those particular parameters, as they were the ones selected during the implementation.

In any case, our performance evaluation of cryptographic operations produced results that can be backed up by theory and that are overall similar to broader performance evaluations of cryptographic algorithms §2.4.3.

Particularly, key generation and signing is faster with ECDSA than with RSA. On the other hand, signature verification operations are faster with RSA than with ECDSA.

Key generation Keys are only generated when the user logs in into a workstation and they last until the session ends or until they expire. Thus, it does not affect the latency of each message.

As a result, the main implication of key generation is their user experience when they log in. Should ECDSA be the algorithm of choice, key generation would be completed in less than a quarter of millisecond in most cases. However, even when using RSA, the median for key generation time is under 9.5 milliseconds, less than a tenth of the maximum time for humans to consider an interaction not to be instantaneous (100 milliseconds) [66].

Moreover, key generation times are practically negligible when operator workstations and verifying nodes do not share the same controller. In those cases, the time since users log in until they are able to log in until they have is affected by the delay of public key propagation among domain controllers.

Since key propagation times are always above 5 seconds, the difference between ECDSA and RSA for key generation would not be appreciable, as this process represent a very small fraction of the total time needed for the log in operation to be completed.

Signature generation and verification Conversely, since each command sent by the user must be signed, signature generation affects the latency of each message. Particularly, RSA signature generation is almost three times slower than its ECDSA counterpart.

On the other hand, RSA is faster performing signature verification than ECDSA. On average, measurements for this operation with ECDSA were approximately 700 times greater than RSA ones. Other studies of cryptographic algorithms [7] also show a better performance in signature verification with RSA than with ECDSA. However, differences are not as great as in our implementation (RSA is between 10 and 50 times faster depending on the particular machine running the tests). Thus, it is likely that our results for ECDSA can be improved by using a more efficient implementation of its verification algorithm.

Message overhead During the evaluation, the overhead of our solution for each message has been studied. The size of this overhead varies from 334 to 522 bytes depending on the algorithm in place. These values can be considered negligible for nowadays the standards (the majority of broadband connections in Sweden have at least a 100 Mbit/s bandwidth [67]).

In any case, these overhead measurements can be shrunk by using binary based formats, instead of JSON. This would help not only to reduce the bandwidth used for message exchange, but also to decrease the processing effort required for converting the data to and from JSON.

Recommendations Signature generation and verification are the measurements to consider when deciding which algorithm is best in terms of latency per message, because key generation does not have an effect on every message. From those results, it is clear that RSA is better than ECDSA, since the difference between ECDSA and RSA for signature verification is far more significant than the variation between those algorithms for signature generation.

Using ECDSA would be appropriate if operator workstations are very lightweight nodes and its is more convenient for the organization to move computational intensive operations to servers. Moreover, ECDSA implementation may be improved by using other libraries or more efficient programming languages.

In addition, RSA key material is longer than the one used by ECDSA. Thus, the size overhead of ECDSA is smaller than that of RSA. Nevertheless, as previously stated, the overhead can be considered as negligible for modern connections. Switching to ECDSA for this reason would only be necessary for very bandwidth constraint connections or in case of limited size fields in the protocols that will transfer the signature data for each command.

Finally, should vulnerabilities be found in RSA, switching to ECDSA would be a good alternative for avoiding them and maintaining the security of the system.

10.3.2 Key retrieval delay

When users log in, their keys are uploaded to their AD domain controller. However, if many controllers are in place, and the verifying node is connected to

a different controller, public keys must be propagated among controllers before the receiving server can verify messages.

Particularly, if controllers are on the same site, the default synchronization delay is 15 seconds. However, in the case of controllers in different sites, the default replication time is 180 minutes. It can be reduced in the site link settings, but the minimum delay is 15 minutes, which is not acceptable for our solution to work properly.

The *USE_NOTIFY* option reduces this time to 15 seconds, by sending notifications for each modification on the AD to all the controllers on the site link. Nevertheless, this approach is very coarse-grained (i.e. it triggers a replication not only for public key field modifications, but for any change on the AD).

Thus, a finer-grained solution has been implemented. It consists in event-triggered synchronization of the user keys (§8.1.5). When the public key field of a user changes, the related AD object is synchronized. In environments with several controllers, it provides faster replication than the rest of studied options, performing it in less than 7 seconds. Particularly, replication latency was 6 ± 1 seconds after system boot, which is likely related with modules being loaded into the system. However, after the initial execution of the synchronization script, the delay in subsequent runs is reduced to 4 ± 1 seconds.

Both values fit ABB requirement for login in times, which establish a maximum of 10 seconds. Moreover, for guaranteeing the availability of the services they provide, domain controllers are up most of the time, being boot operations seldom. Therefore, most executions of the event-triggered task would be completed in approximately 4 seconds.

As a result, should controllers be in different logical sites, event-triggered AD synchronization is clearly the best choice, since replication is only performed when a public key field is modified and the delay is lower than with the *USE_NOTIFY* parameter. Thus, the logging in latency is lower, and the bandwidth utilization of the link between sites is also minimized.

Furthermore, event-triggered AD synchronization is also faster than default replication times between controllers in the same site. As a result, even though the difference is not as significant as with controllers in different sites, it could be beneficial to apply this solution in that case, in order to fit ABB login delay requirements.

In summary, if signing and verifying nodes share the same controller, key retrieval delay is only dependant on the network delay. Otherwise, the additional delay produced by AD replication will also affect key retrieval times. Particularly, the minimum retrieval time in that case 4 ± 1.

Since login only happens once for each session, we consider the delay to be acceptable. The login process is performed at the beginning of the session. Hence, although it creates a delay for command issuing capabilities to be available to the operator, it does not interrupt or increment the latency of individual messages throughout the session.

11 Conclusion

The aim of this Master thesis was to provide an answer to the following research question:

> What is the best method to provide authentication and accountability on the communication between a SCADA server and a remote operator when an access server is in the middle?

The design proposed in this thesis provides an answer to that question, by delivering a solution that solves the stated problem. Moreover, two aspects of the research question were considered: a theoretical view, whose solely focus was the security of the proposed solution; and a practical view, that additionally considered other aspects such as cost, latency and compatibility with the target architecture.

The final solution simultaneously addresses both aspects of the research question. Firstly, it is based on digital signatures, that provide the higher level of security of all the options considered in the analysis (§6), consequently our design provides an answer to the research question from a theoretical view. Secondly, the proposed solution also addresses the practical side of the research question, since the design has also been evaluated in terms of latency, implementation cost and compatibility with the existing architecture; guaranteeing its suitability.

Furthermore, the solution proposed in the design chapter (§7) of this Master thesis has been validated by creating a prototype implementation (§8) and fulfils the requirements established in §5, in terms of compatibility with the target architecture, security and performance.

Particularly, a high level of security is achieved by the use of asymmetric cryptography. However, for reducing implementation and management costs, we do not rely on a Public Key Infrastructure (PKI) for credential management. Conversely, we benefit from the presence of Microsoft Active Directory on the target architecture. We link its identities with the short term public keys that identify the issuer of each digital signature.

11.1 Future work

The objective of this Master thesis has been met. However, it may be extended in future work. Particularly, the implementation may be improved by adjusting it to work on top of ABB RPC protocols. This would allow ABB using our design in production environments and would provide a more reliable test bed for performance evaluations.

Moreover, efficiency improvements can be applied to the implementation. Firstly, switching from Python to C/C++ would improve the performance at the verifying node, as stated in §8.3. In addition, moving from JSON to a custom binary format for exchanging messages between nodes would reduce the bandwidth used by our solution and improve performance, as JSON parsing operations would not be needed any more.

What is more, controller synchronization times when signing and verifying nodes do not share the same controller have been limited by event triggered synchronization to acceptable values. Nevertheless, further reductions of this delay would be beneficial for the experience provided to the operators when issuing commands.

Furthermore, an alternative implementation that does not rely on Active Directory for credential management, but on alternative technology, may be proposed. For instance, a SQL server could be used for storing public keys. Having an alternative implementation would broaden the possible target architectures of our solution, as a Microsoft AD infrastructure would no longer be needed.

In addition, neither RSA nor ECDSA are supposed to be resistant to quantum computing (§2.4.4). Thus, alternative implementations that rely on quantum-resistant algorithms (e.g lattice-based or hash-based cryptography) are another possible extension of this work, as they would make this design valid even if, due to the development of quantum computing, RSA and ECDSA become vulnerable.

References

[1] Phayzfaustyn, "Symmetric key encryption," image. [Online]. Available: https://en.wikipedia.org/wiki/File:Symmetric_key_encryption.svg

[2] Davidgothberg, "Public key encryption," image. [Online]. Available: https://en.wikipedia.org/wiki/File:Public_key_encryption.svg

[3] D. Sonck, "Kerberos negotiations," image, 2011. [Online]. Available: https://commons.wikimedia.org/wiki/File:Kerberos.svg

[4] (2004) Design science research in information systems. [Online]. Available: http://www.desrist.org/design-research-in-information-systems/

[5] E. Barker, *SP 800-57 Part 1 Rev. 4. Recommendation for Key Management, Part 1: General*, NIST, 2016. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-57-part-1/rev-4/final

[6] ECRYPT II. European Network of Excellence in Cryptology II, "ECRYPT II Yearly Report on Algorithms and Keysizes," 2012. [Online]. Available: http://www.ecrypt.eu.org/ecrypt2/

[7] ECRYPT II. (2017) eBACS: ECRYPT Benchmarking of Cryptographic Systems. [Online]. Available: https://bench.cr.yp.to/results-sign.html

[8] ECRYPT II . (2017) Implementation notes: amd64, genji262, crypto_sign. [Online]. Available: https://bench.cr.yp.to/web-impl/amd64-genji262-crypto_sign.html

[9] W. Stallings, *Network Security Essentials: Applications and Standards*, 4th ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2010.

[10] E. Barker, *FIPS 186-4. Digital Signature Standard (DSS)*, US National Institute of Standards and Technology, 2013. [Online]. Available: https://csrc.nist.gov/publications/detail/fips/186/4/final

[11] National Security Agency, *CNSA Suite and Quantum Computing FAQ*, 2016. [Online]. Available: https://www.iad.gov/iad/library/ia-guidance/ia-solutions-for-classified/algorithm-guidance/cnsa-suite-and-quantum-computing-faq.cfm

[12] H. Delfs and H. Knebl, *Introduction to Cryptography: Principles and Applications*, 3rd ed., 2015.

[13] S. Vaudenay, *A Classical Introduction to Cryptography*, 2006.

[14] N. Jansma and B. Arrendondo, "Performance Comparison of Elliptic Curve and RSA Digital Signatures," 2004. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.129.7139&rep=rep1&type=pdf

[15] K. M. Martin, *Everyday Cryptography: Fundamental Principles and Applications*, 2013.

[16] Symantec, "Elliptic Curve Cryptography (ECC) Certificates Performance Analysis," 2013. [Online]. Available: https://www.websecurity.symantec.com/content/dam/websitesecurity/digitalassets/desktop/pdfs/whitepaper/Elliptic_Curve_Cryptography_ECC_WP_en_us.pdf

[17] V. Gupta, S. Gupta, S. Chang, and D. Stebila, "Performance Analysis of Elliptic Curve Cryptography for SSL," in *Proceedings of the 1st ACM Workshop on Wireless Security*, ser. WiSE '02. New York, NY, USA: ACM, 2002, pp. 87–94. [Online]. Available: http://doi.acm.org.focus.lib.kth.se/10.1145/570681.570691

[18] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1484–1509, 1997. [Online]. Available: https://doi.org/10.1137/S0097539795293172

[19] D. J. Bernstein, *Introduction to post-quantum cryptography*, 2009.

[20] Thorsten Kleinjung et al., "Factorization of a 768-bit RSA modulus," 2010. [Online]. Available: https://eprint.iacr.org/2010/006.pdf

[21] Arjen K. Lenstra et al., "Ron was wrong, Whit is right," 2012. [Online]. Available: https://eprint.iacr.org/2012/064.pdf

[22] Matus Nemec et al., "The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli," 2017. [Online]. Available: https://crocs.fi.muni.cz/_media/public/papers/nemec_roca_ccs17_preprint.pdf

[23] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," 1995. [Online]. Available: https://www.paulkocher.com/TimingAttacks.pdf

[24] D. J. Bernstein. (2014) How to design an elliptic-curve signature system.

[25] Alfred J. Menezes et al., *Handbook of Applied Cryptography*, 1996.

[26] N. Lawson. (2011) DSA requirements for random k value. [Online]. Available: https://rdist.root.org/2010/11/19/dsa-requirements-for-random-k-value/

[27] T. Pornin. (2013) RFC 6979: Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA). [Online]. Available: https://tools.ietf.org/html/rfc6979

[28] H. Delfs and H. Knebl, *Introduction to Cryptography: Principles and Applications*, 3rd ed., 2015.

[29] D. J. Bernstein and T. Lange. (2013) SafeCurves: choosing safe curves for elliptic-curve cryptography - Rigidity. [Online]. Available: https://safecurves.cr.yp.to/rigid.html

[30] A. Jivsov. (2012) Elliptic Curve Cryptography (ECC) in OpenPGP.

[31] J. Katz and Y. Lindell, *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[32] *NIST Brief Comments on Recent Cryptanalytic Attacks on Secure Hashing Functions and the Continued Security Provided by SHA-1*, NIST, 2004. [Online]. Available: https://csrc.nist.gov/News/2004/NIST-Brief-Comments-on-Recent-Cryptanalytic-Attack

[33] Marc Stevens et al. (2017) Announcing the first SHA1 collision. [Online]. Available: https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html

[34] *SHA-3 Project*, NIST, 2017. [Online]. Available: https://csrc.nist.gov/Projects/Hash-Functions/SHA-3-Project

[35] *NIST Policy on Hash Functions*, NIST, 2015. [Online]. Available: https://csrc.nist.gov/Projects/Hash-Functions/NIST-Policy-on-Hash-Functions

[36] H. Krawczyk et al. (1997) Rfc 2104: Hmac: Keyed-hashing for message authentication. [Online]. Available: https://tools.ietf.org/html/rfc2104

[37] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication." Springer-Verlag, 1996.

[38] S. Turner et al. (2011) Rfc 6151: Updated security considerations for the md5 message-digest and the hmac-md5 algorithms. [Online]. Available: https://tools.ietf.org/html/rfc6151

[39] C. Neuman et al. (2005) Rfc 4120: The kerberos network authentication service (v5). [Online]. Available: https://tools.ietf.org/html/rfc4120

[40] K. Raeburn. (2005) Rfc 3961: Encryption and checksum specifications for kerberos 5. [Online]. Available: https://tools.ietf.org/html/rfc3961

[41] K. O. Yu Sasaki, Lei Wang and N. Kunihiro. (2007) New message difference for md4. [Online]. Available: https://www.iacr.org/archive/fse2007/45930331/45930331.pdf

[42] *[MS-KILE]: Kerberos Protocol Extensions*, Microsoft, 2017. [Online]. Available: https://msdn.microsoft.com/en-us/library/cc233855.aspx

[43] S. Duckwall and B. Delpy. (2014) Abusing kerberos. [Online]. Available: https://www.blackhat.com/docs/us-14/materials/us-14-Duckwall-Abusing-Microsoft-Kerberos-Sorry-You-Guys-Don't-Get-It-wp.pdf

[44] G. Kiwi. (2014) Pass the ticket. [Online]. Available: http://blog.gentilkiwi.com/securite/mimikatz/pass-the-ticket-kerberos

[45] P. Nygaard. (2011) Kerberos delegation. [Online]. Available: https://blogs.msdn.microsoft.com/autz_auth_stuff/2011/05/03/kerberos-delegation/

[46] K. S. Keith Stouffer, Joe Falco, *Guide to Industrial Control Systems (ICS) Security*, 2011.

[47] E. J. M. Colbert and Alexander Kott, *Cyber-security of SCADA and Other Industrial Control Systems*, 2016.

[48] Primatech, *FAQ Sheet S84 / IEC 61511 Standard for Safety Instrumented Systems*, 2009. [Online]. Available: http://www.primatech.com/images/docs/faq_s84_standard_for_safety_instrumented_systems.pdf

[49] M. S. Thomas and J. D. McDonald, *Power System SCADA and Smart Grids*, 2016.

[50] J. Luque, J. I. Escudero, and F. Perez, "Analytic model of the measurement errors caused by communications delay," *IEEE Transactions on Power Delivery*, vol. 17, no. 2, pp. 334–337, Apr 2002.

[51] K. L. I. CERT, "Threat landscape for industrial automation systems in the second half of 2016," 2017. [Online]. Available: https://ics-cert.kaspersky.com/wp-content/uploads/sites/6/2017/03/KL-ICS-CERT_H2-2016_report_FINAL_EN.pdf

[52] I. M. S. Services, "Security attacks on industrial control systems: How technology advances create risks for industrial organizations," 2015. [Online]. Available: https://securityintelligence.com/media/security-attacks-on-industrial-control-systems/

[53] ICS-CERT. Standards and references. [Online]. Available: https://ics-cert.us-cert.gov/Standards-and-References

[54] ENISA. (2015) Cyber security information sharing: An overview of regulatory and non-regulatory approaches. [Online]. Available: https://www.enisa.europa.eu/publications/cybersecurity-information-sharing

[55] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," 2004. [Online]. Available: http://www.jstor.org/stable/25148625

[56] R. Budde, K. Kuhlenkamp, K. Kautz, and H. Zulighoven, *Prototyping: An Approach to Evolutionary System Development*. Berlin, Heidelberg: Springer-Verlag, 1992.

[57] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45–77, 2007. [Online]. Available: https://www.tandfonline.com/doi/abs/10.2753/MIS0742-1222240302

[58] J. Walton. (2018) Crypto++ 6.0.0 benchmarks. [Online]. Available: https://www.cryptopp.com/benchmarks.html

[59] V. Malhotra. (2008) Windows administration: Extending the active directory schema. [Online]. Available: https://technet.microsoft.com/en-us/library/2008.05.schema.aspx

[60] E. Barker. (2013) FIPS 186-4: Digital Signature Standard (DSS). [Online]. Available: https://csrc.nist.gov/publications/detail/fips/186/4/final

[61] C. Duffey. (2012) Active directory replication: Change notification & you. [Online]. Available: https://blogs.msdn.microsoft.com/canberrapfe/2012/03/25/active-directory-replication-change-notification-you/

[62] T. Salmon. (2016) Storing ssh keys in active directory for easy deployment. [Online]. Available: https://blog.laslabs.com/2016/08/storing-ssh-keys-in-active-directory/

[63] Balabit. (2015) Ssh usermapping and keymapping in ad with public key. [Online]. Available: https://www.balabit.com/documents/scb-latest-guides/en/scb-guide-admin/html/proc-scenario-usermapping.html

[64] Derek. (2015) Authorized_keys in active directory. [Online]. Available: http://www.theendofthetunnel.org/2015/11/21/authorized_keys-in-active-directory/

[65] R. Ries. (2014) Store pgp keys on active directory/exchange? [Online]. Available: https://serverfault.com/a/610020

[66] J. Nielsen, *Usability engineering*, ser. Interactive Technologies. Cambridge, Mass.: AP Professional, 1993.

[67] K. Fransén and A. Wigren, "Svensk telekommarknad första halvåret 2017," 2017. [Online]. Available: http://www.statistik.pts.se/media/1266/stm-1h2017_slut.pdf

Glossary

Kerberos Network based authentication protocol. It relies on symmetric key cryptography and the use of encrypted tickets.

nonce Arbitrary number that can only be used once.

RPC Inter-process communation technology that provides location transparency. Calls to remote procedures can be done as if they where local.

RSA Asymmetric key algorithm for digital signature and encryption.

SCADA Computer based system architecture for management and supervision of industrial processes.

service user User account that does not represent a user, but a service; providing a security context for the service execution.

Acronyms

AD Active Directory.

ANSI American National Standards Institute.

AS Authentication Server.

BAS Building Automation System.

CA Certification Authority.

CRL Certificate Revocation List.

DCS Distributed Control System.

DoS Denegation of Service.

DSA Digital Signature Algorithm.

ECC Elliptic Curve Cryptography.

ECDSA Elliptic Curve Digital Signature Algorithm.

EMS Energy Management System.

ENISA European Union Agency for Network and Information Security.

HMAC keyed-Hash Message Authentication Code.

ICS Industrial Control System.

IED Intelligent Electronic Device.

KDC Key Distribution Center.

LDAP Lightweight Directory Access Protocol.

MAC Message Authentication Code.

MD5 Message Digest 5.

MITM man-in-the-middle.

NIST United States National Institute of Standards and Technology.

NSA United States National Security Agency.

NTLM Microsoft NT LAN Manager.

OCSP Online Certificate Status Protocol.

OID Object IDentifier.

OU Organizational Unit.

PCS Process Control System.

PKI Public Key Infrastructure.

PLC Programmable Logical Controller.

RFC Request For Comments.

RPC Remote Procedure Call.

RSA Rivest, Shamir and Adleman.

RTT Round Trip Time.

RTU Remote Terminal Unit.

SCADA Supervisory Control And Data Acquisition.

SHA Secure Hash Algorithm.

SIS Safety Instrumented System.

SPOF Single Point Of Failure.

SQL Structured Query Language.

TGS Ticket Granting Server.

TGT Ticket Granting Ticket.

WAN Wide Area Network.

A Installation manual for the Active Directory Domain Controller

This appendix provides guiding for administrators to set up a domain controller to work with the solution proposed in this thesis.

Adding a field for storing operator keys

An Object IDentifier (OID) is needed for creating attributes and classes on the AD schema. They are unique identifiers for each element. In our mock implementation we used a base OID obtained from Microsoft by running the *oidgen.vbs* script. This script was obtained from Microsoft's TechNet gallery and is included with the rest of our implementation.

Furthermore, following a common schema structure, we will have classes under *BaseOID*.1 and attributes under *BaseOID*.2. However, this is by no means mandatory: any extension schema may be used.

When deploying this solution in a production environment, a different base OID must be used. It may be requested from an ISO member for the owning organization, as explained in `https://technet.microsoft.com/en-us/library/2008.05.schema.aspx`. Should that not be possible, one issued by Microsoft through the provided script would also be valid.

Once a base OID is available, a field for storing public keys in the user class must be added to the Active Directory. A Powershell Script (*ADSchemaUserPublicKey.ps1*) for performing that operation is provided. Particularly, it performs the following actions:

1. It enables AD Schema modifications on the domain controller running the script.
2. It creates an ldif script that:
    2.1. Creates an attribute for storing public keys.
    2.2. Creates an auxiliary operator class for holding the attribute.
    2.3. Sets the created class as auxiliary of the User class.
3. It runs the created ldif script.

The scripts needs some parameters in order to create a custom *ldif* script, depending on the naming policies of the organization. Table **??** lists the different parameters accepted by the script.

Setting basic permissions

Once the script has been run correctly, the attribute for storing public keys will be in the Active Directory schema, belonging to the *user* class. The name of the attribute will be *prefix*OperatorPublicKeys. However, the attribute must have the appropriate permissions, as defined in the design chapter.

Name	Required	Description
prefix	Yes	A common prefix for the attribute and class to be created. Custom Active Directory elements should have a common prefix. Should start with a lower case letter an written in *camelCase*.
baseOID	Yes	The base OID for the organization owning the added elements. It should be related with the Prefix.
publicKeyAttributeOID	No (Default: "2.1")	The OID for the public Key Attribute. It is appended to the base OID for identifying the new element.
operatorClassOID	No (Default: "1.1")	The OID for the operator class. It is appended to the base OID for identifying the new element.

Table 12: Public Keys script parameters

By default, any domain user can read the new field, but it can only be written by domain administrators. In order to provide writing permissions for each user, domain administrators must delegate this privilege to users. The steps for achieving this are detailed below:

1. Open the Active Directory User and Computers tools. Can be invoked from the Run Command tool (Windows Key + R), by entering "dsa.msc".
2. Open the *Delegation of Control Wizard* for the User container that contains the users who need to store their public keys. Figure 17 shows how to open the wizard for the User container of the *test.m* domain. The following nested list details what to do in each step of the wizard:
   - 2.1. Informational text, no action needed.
   - 2.2. **User or groups:** User *SELF* must be added to the list, as Figure 18 shows.
   - 2.3. **Tasks to Delegate:** Select the second option "Create a custom task to delegate" (Figure 19).
   - 2.4. **Active Directory Object Type:** Select only the user object (Figure 20).
   - 2.5. **Permissions:** Select the *Property-specific* box on the top list. Then, on the box, enable the permissions for reading and writing the *<Prefix>OperatorPublicKeys* field (Figure 21).
   - 2.6. Check that everything in order and press *Finish*

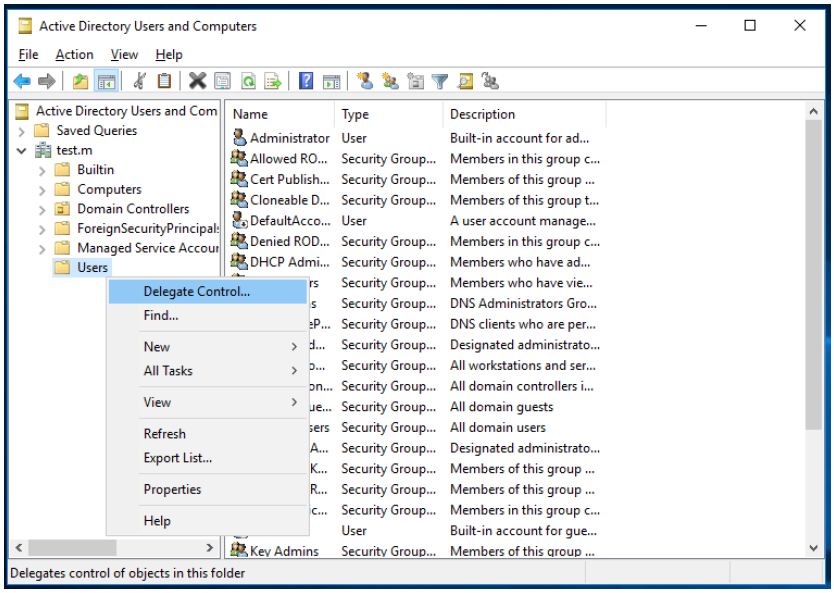

Figure 17: Opening the delegation wizard for the *User* container in the *test.m* domain

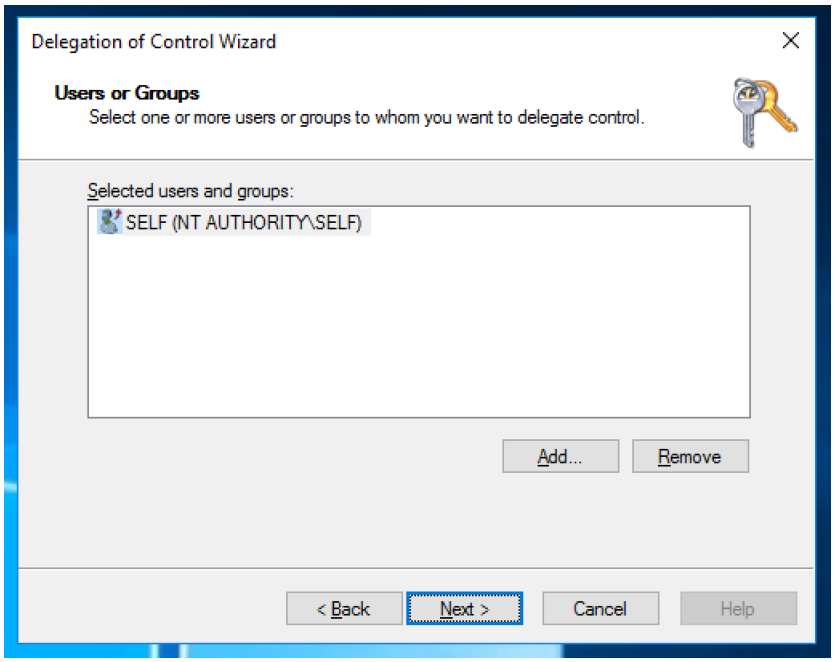

Figure 18: *Delegation of Control Wizard*: *SELF* user added for delegation

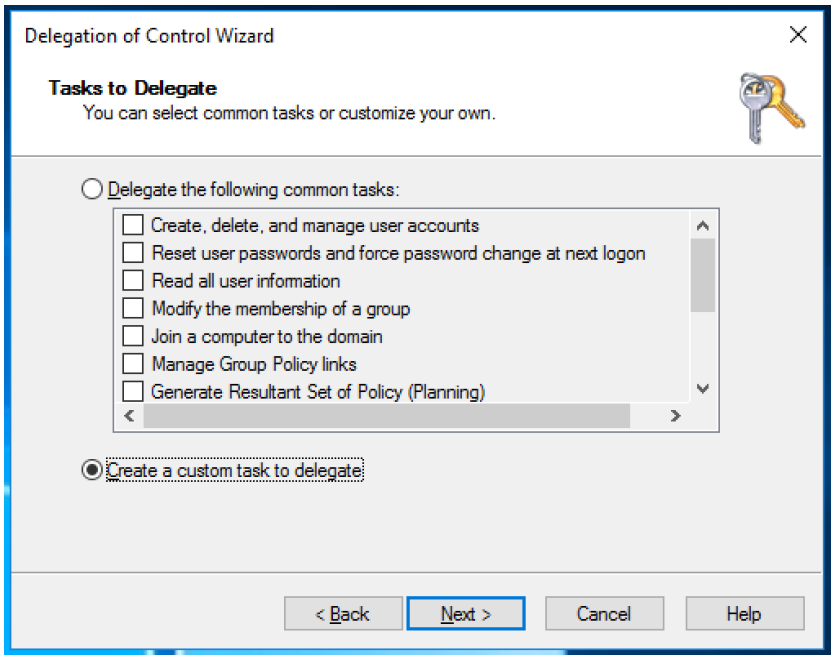

Figure 19: *Delegation of Control Wizard*: Create a custom task to delegate

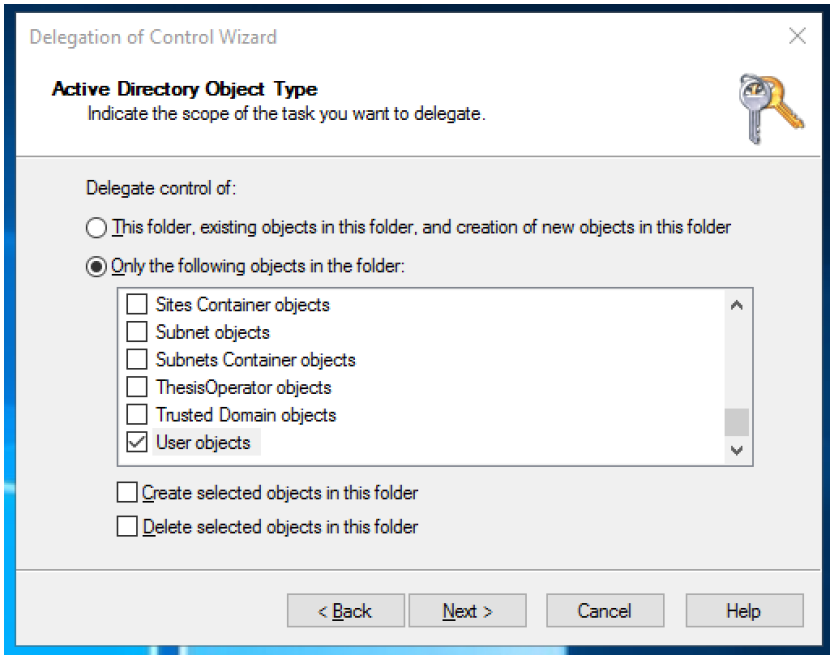

Figure 20: *Delegation of Control Wizard*: Selecting the *User* object

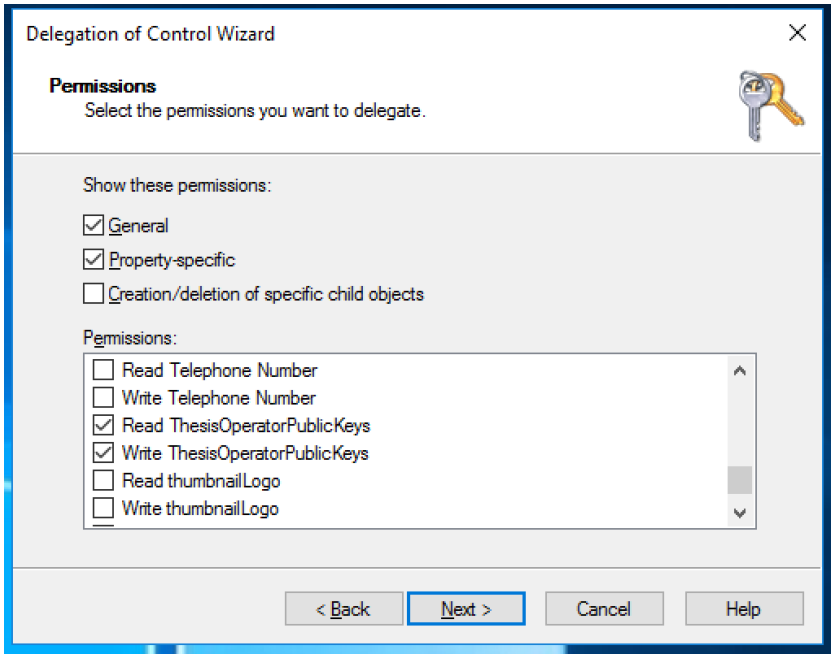

Figure 21: *Delegation of Control Wizard*: Selecting the field that stores the keys

Applying finer-grained policies

After the steps covered in the previous section (§A) have been followed, only the user itself (and domain administrators) would be able to write their own public key into the Active Directory. Reading permissions are granted to all authenticated users.

Firstly, a policy that allows some users (service accounts) to read the field that stores the public keys must be implemented. It could be done group by group if more fine-grained policies are needed. However, for simplicity, we are using the *Users* container of our domain:

- Open the Active Directory User and Computers tools
- Enable the *Advanced Features* from the *View* Menu
- Open the properties of the *Users* container inside your domain.
- Go to the *Security* tab and press the *Advanced* button (Figure 22).
- Add a new rule with the following configuration (Figures 23 and 24):
    1. Principal: The group that contains the service accounts which are authorised to read the keys
    2. Type: *Allow*
    3. Applies to: *Descendant User objects*
    4. Only one checkbox must be selected: *Read <prefix>OperatorPublicKeys*
- Press OK and apply the changes

Moreover, for granting reading permissions only to particular users (service accounts and owner user), a *default deny* policy must be implemented. It should block reading requests from every user in the domain (administrators will not be affected by this policy). As discussed in the implementation chapter (§8), this policy must be implemented higher element in the AD hierarchy than the previous rule. Since our mock implementation does not have any OUs, we implemented the rule domain-wise:

- Still from the Active Directory User and Computers tools (with the *Advanced Features* enabled), open the properties for the domain.
- Go to the *Security* tab and press the *Advanced* button.
- Add a new rule with the following configuration (Figures 25 and 26):
    1. Principal: The group that contains the service accounts which are authorised to read the keys
    2. Type: *Allow*
    3. Applies to: *Descendant User objects*
    4. Two checkboxes must be selected: *Read <prefix>OperatorPublicKeys* and *Write <prefix>OperatorPublicKeys*. The latter is not strictly necessary but guarantees security if the schema is changed later.
- Press OK and apply the changes

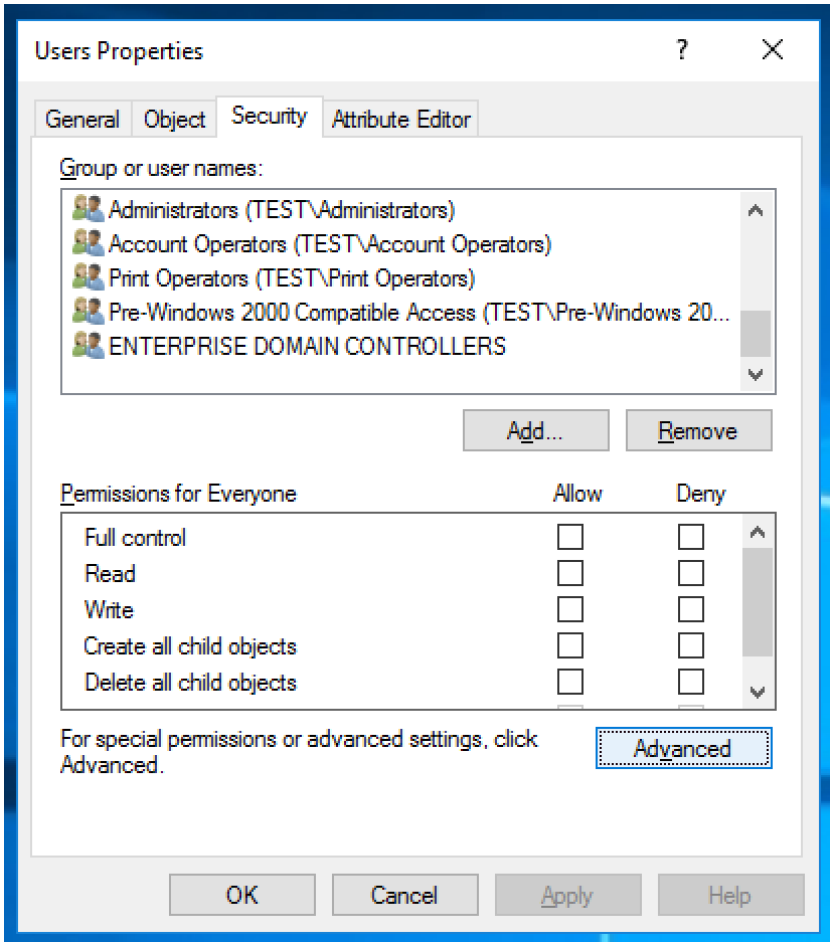

Figure 22: Advanced security options the *Users* container in the *test.m* domain

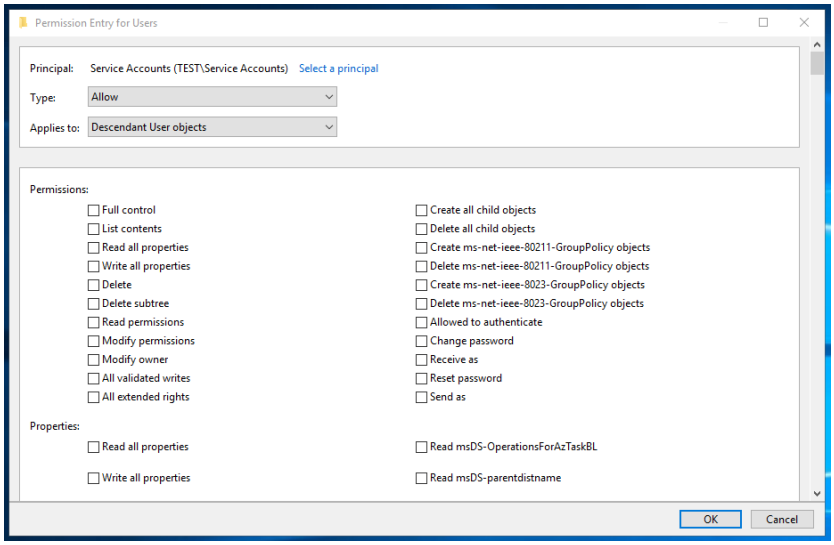

Figure 23: *Users* container *allow* policy

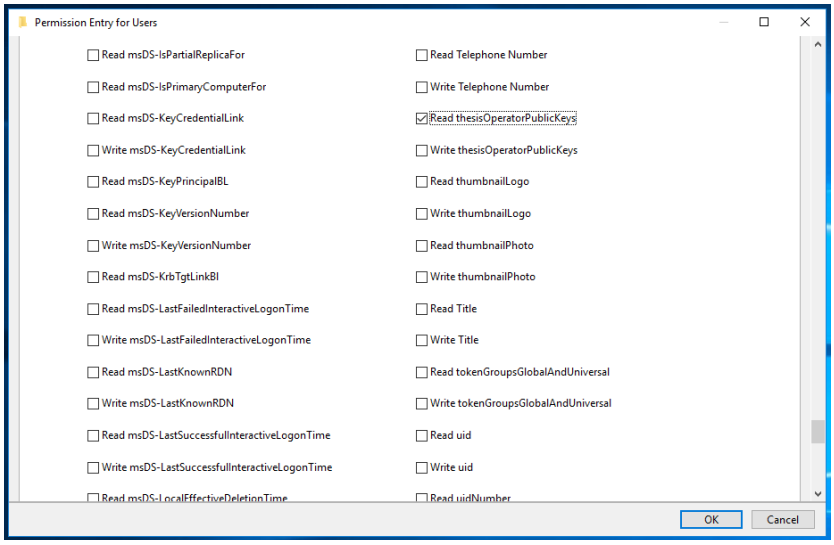

Figure 24: *Users* container *Read thesisOperatorPublicKeys*

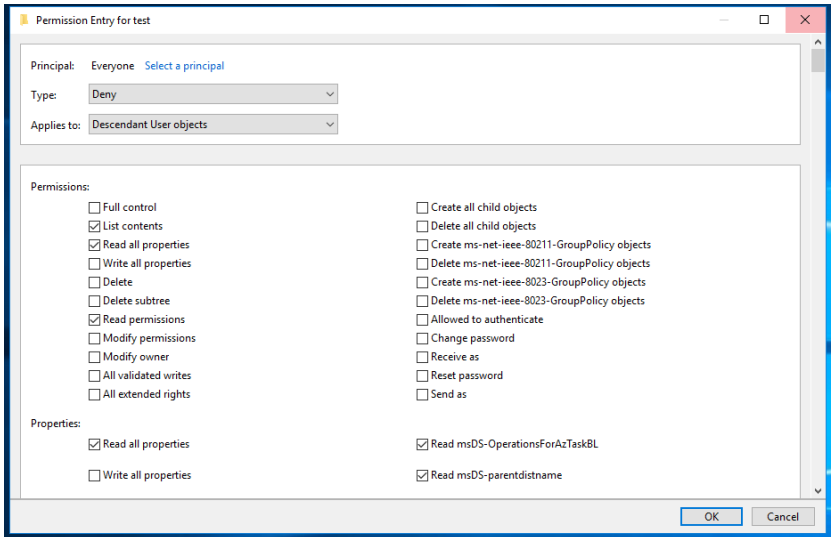

Figure 25: *test* domain container *deny* policy

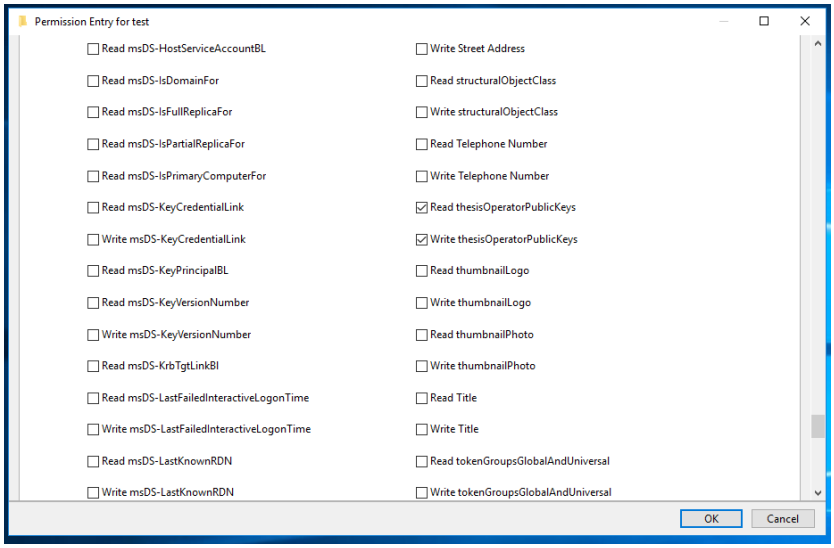

Figure 26: *test* domain read and write *thesisOperatorPublicKeys*

Purging obsolete keys

As discussed during the implementation section, expired keys may stay in the Active Directory if sessions are not closed graciously. In order to address this issue, the script *CleanExpiredPublicKeys.ps1* is provided. When run from a domain controller, it removes all expired public keys.

In order to automatize the operation, a task can be created with the Windows Task Scheduler (*taskschd.msc*), using the creation task Action. The following parameters are recommended:

- In the General tab, change the user to *NT AUTHORITY\SYSTEM* (it can be selected just by writing *SYSTEM* on the *Select User or Group* dialog box)
- In the Triggers tab, add a time based trigger. The frequency can be set depending on the requirements of the particular deployment
- In the Actions tab, execute the *Add new task* action:
  - Program/script: powershell
  - Add arguments: -File *path_to_CleanExpiredPublicKeys.ps1*

Domain controller replication

As discussed in the implementation chapter, two options have been explored for dealing with the latency of domain controller replication: Enabling notifications for all changes in the Active Directory and a finer-grained approach based on the logging of AD write operations.

***USE_NOTIFY* option**

This solution for avoiding replication delays only requires the modification of one parameter. In the Active Directory Sites and Services (*dssite.msc*), open the properties for the link that connects your domain controllers (Figure 27). On the Attribute Editor tab, select the *options* attribute an set its value to 1 (Figure 28).

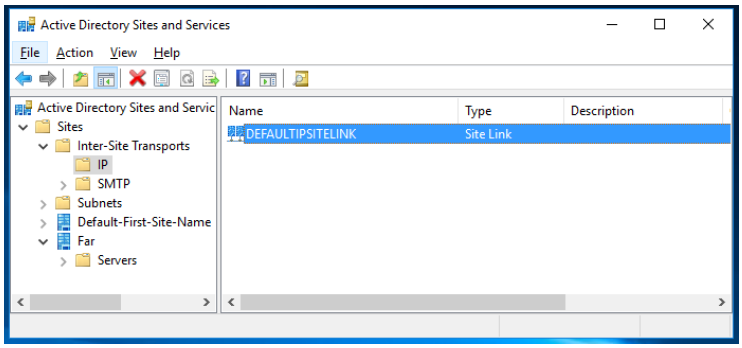

Figure 27: Active Directory Sites and Services IP link

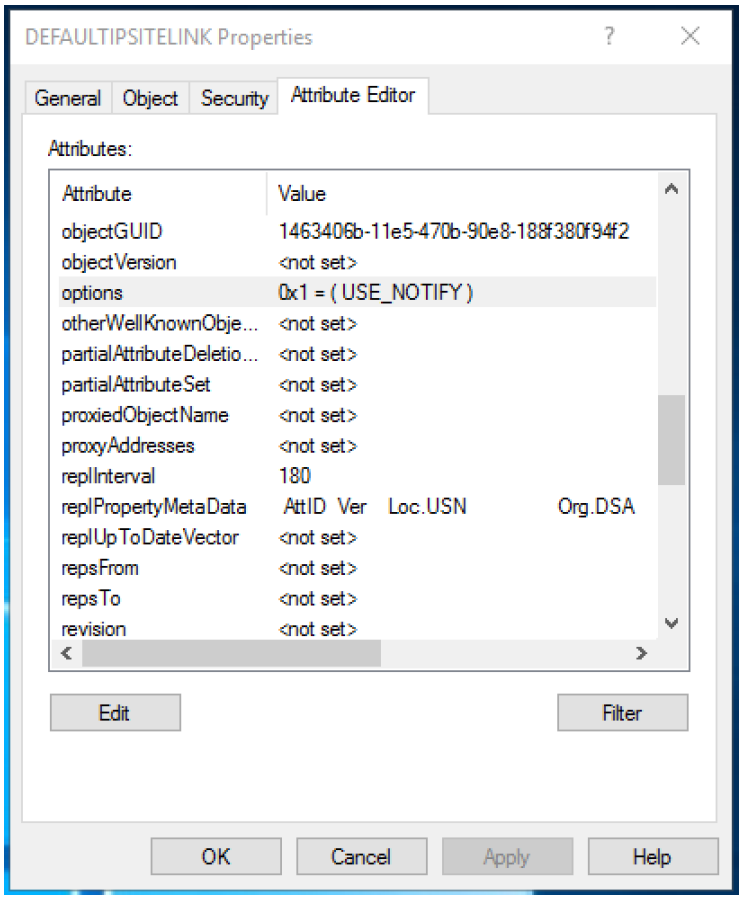

Figure 28: Active Directory Sites and Services: Switching *USE_NOTIFY* on

Logging based solution

In order to use this solution, auditing must be enabled for write accesses to the public-key attribute of the user class. Auditing rules can be established from the Active Directory User and Computers tools (*dsa.msc*), in the Advanced Security Settings for the User container.

On the Auditing tab (Figure 29), a new rule must be added with the following parameters:

1. Principal: Everyone
2. Type: Success
3. Applies to: Descendant User objects
4. Only write thesisOperatorPublicKeys must be selected

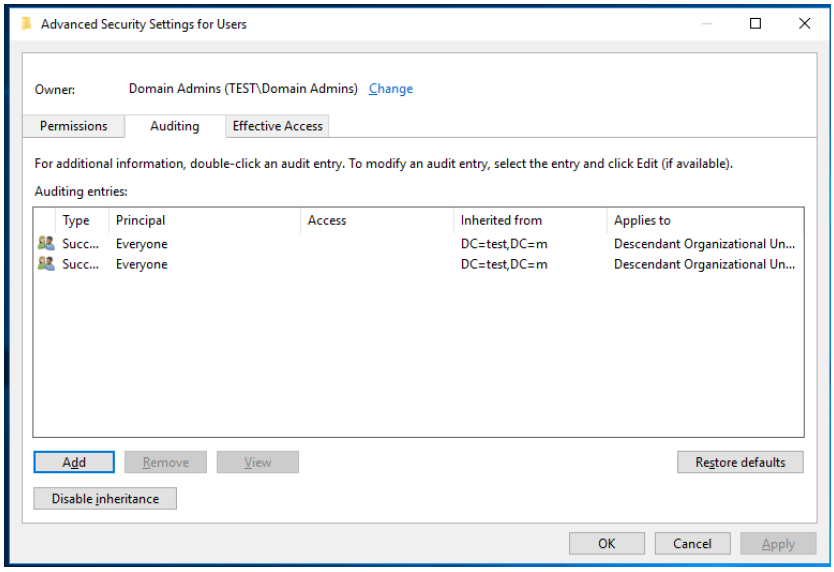

Figure 29: Active Directory User and Computers: User container auditing rules

Once the rule is established, write operations on the public-key field will generate an event. For acting on that event, the *SyncADUserIfKeyUpdated.ps1* script must be stored on a stable location in the domain controller. Then a task triggered by write access to User objects may be created using the *SyncTaskGenerator.ps1* script, that accepts three parameters:

1. **attributename**: The name of the attribute where public keys are stored in the user objects
2. **domainPath**: The path of the domain (i.e. "DC=test,DC=m" for domain test.m)
3. **syncScriptPath**: The path of *SyncADUserIfKeyUpdated.ps1*

After running *SyncTaskGenerator.ps1* the system should automatically perform a replication of users whose public-key attribute is written.